https://escholarship.org/uc/item/6fv1s464
pp. 249–260 in S. Seagroves, A. Barnes, A.J. Metevier, J. Porter, & L. Hunter (Eds.), *Leaders in effective and inclusive STEM: Twenty years of the Institute for Scientist & Engineer Educators*. UC Santa Cruz: Institute for Scientist & Engineer Educators.
https://escholarship.org/uc/isee_pdp20yr

An Inquiry Approach to Teaching Sustainable Software Development with Collaborative Version Control

Rachel L.S. Frisbie[*1], Philipp Grete[2,3], and Forrest W. Glines[1,3]

[1] Department of Computational Mathematics, Science & Engineering, Michigan State University, East Lansing, MI, USA
[2] Hamburg Observatory, University of Hamburg, Hamburg, Germany
[3] Department of Physics & Astronomy, Michigan State University, East Lansing, MI, USA
[*] Corresponding author, salmonra@msu.edu

Abstract

Software development is becoming increasingly ubiquitous in STEM disciplines resulting in the need for education in associated computational skills. To address this need, we designed a "Sustainable Software Development with Collaborative Version Control" workshop in the 2019 Institute for Scientist & Engineer Educators (ISEE) Professional Development Program (PDP). We describe here the development process and following delivery of the workshop. In particular, we explored how to apply an inquiry approach to learning computational skills. By design, PDP activities intertwine content and "cognitive STEM practices," and teasing apart content and practice is important for STEM education. We encountered challenges with this task because our content — exploring software sustainability with collaborative version control — is much like a practice in itself. We designed our workshop to introduce the critical skill of sustainable software development using collaborative version control systems with an inquiry approach rather than the more typically used, strictly technical approach. We emphasize the authentic, broadly applicable nature of the workshop in which learners jointly design, test, and discuss their own increasingly complex development workflows. The development process for our workshop may be useful for educators who want to introduce software practices to learners from many disparate STEM disciplines that leverage computational methods and require software development to approach research questions.

Keywords: activity design, git, inquiry, version control, software development

1. Introduction

The ultimate goal of our workshop was to help learners learn concepts that would enable sustainable development of scientific software. As computing resources become more intrinsic to scientific research, more scientists are developing software to enable their research. These software projects may range from small scripts used to analyze experimental data, to specialized software used to control instrumentation, to large simulation codes used to model physical systems. Development of scientific

software may be carried out by groups as small as a single scientist or engineer, to mentor-mentee pairs, to research groups under a principal investigator, to large collaborations spread across institutions in academia, government, and industry. In any of these cases, software sustainability practices — including but not limited to — tracking and communicating bugs and desired features, tracking and managing changes to the software, and assigning and delegating roles and responsibilities to software developers and users — are majorly beneficial to the reliability, accuracy, and maintainability of scientific software (Nangia et al 2017, Queiroz et al. 2017). Unfortunately, said development practices are not yet widely implemented in scientific software development, hence our motivation to train upcoming scientists and engineers in sustainable software practices.

Development workflows that enact said practices are usually facilitated by internet hosting sites for *version control systems*, software that enables the tracking and management of the source code for software. At the time of writing, the most popular version control system is "git", with github.com and gitlab.com being popular online services that, along with hosting the source code for projects managed with git, enable essential discussion and collaboration of code changes.

At the heart of such a software project is the *source code repository*, or "repo," which is the collection of all source code and the history of changes to the source code. Development of the source code may persist along different routes known as "branches." For example, there may be a "stable" branch of the repo that has been thoroughly tested and an "experimental" branch of the repo where new less tested features are under development. Changes to the code are added to branches in a "commit." A commit refers to a set of changes to one or many files within the source tree of the repository, effectively also specifying a snapshot of the source code. Commits from different branches can be combined via a "merge". Creating and managing branches and commits as well as merging branches can be accomplished locally on a developer's computer either via the command line or graphical tools or via interfaces provided by the internet hosting sites such as gitlab.com. Additionally, the internet hosting sites usually provide discussion boards to make comments on code changes, document bugs, request new features, and any other discussion of the code. Branch mergers are typically accomplished and discussed in "merge requests" on GitLab (or equivalently "pull requests" on GitHub). "Issues" enable further discussion, providing a tool to document and discuss bugs in the source code, request new features, and make other discussions about the repo. These tools within git and the internet hosting sites enable workflows incorporating sustainable software practices.

Abundant literature supports the claim that learning to program can be difficult, and exploring new ways to teach computational concepts can help improve learners' understanding (e.g. Guzdial 2010, 2013, Hazzan et al. 2011, Sorva 2012, Porter et al. 2013). Exploring topics with an Inquiry framework, as in the PDP, can increase learner understanding (Metevier et al. 2022a, 2022b) and help learners build their identities as scientists (Carlone & Johnson 2007). We sought out to design our workshop within the PDP in part to address the need for a more effective way to teach sustainable software practices to early-stage programmers.

2. Workshop overview

2.1 Venue and learners

We developed our workshop, *Sustainable Software Development,* as part of the 2019 PDP. We designed the workshop for learners from the 2019 Michigan State University (MSU) Advanced Computational Research Experience (ACRES) and the 2019 MSU Physics and Astronomy Research Experience for Undergraduates (REU). An REU Site consists of a group of ten or so undergraduates who work in the research programs of the host institution. REU sites

are encouraged to involve students from historically marginalized groups. Each student is associated with a specific research project, where they work closely with the faculty and other researchers (https://www.nsf.gov/crssprgm/reu/). These learners had significant variation in prior knowledge about computation; consequently, we designed our workshop with that in mind. We chose to have learners engage with Gitlab, a web-based git platform, and basic text-based documents to avoid potential software issues and eliminate the need for prerequisite knowledge of the terminal, a specific programming language, and git to be able to engage in the workshop. We ran the workshop twice, first with the group of ACRES students and second with the group of Physics and Astronomy REU students. Our workshop spanned three hours and was split into two sessions with a lunch break in between for both venues. In 2020 and 2021, we adapted this workshop to be run virtually with both REU programs and retained the basic structure from 2019.

Our primary goal for the workshop was to introduce the concepts of sustainable software development using git as a tool. In our experiences, git is typically presented as a list of commands to be used from the terminal while discussion of workflow structure and cases of practical and real-world use is minimal. We set out to create an opportunity for learners to discover for themselves how to develop an effective workflow and then learn the git tools necessary to maintain that workflow. We believed that many of the learners, particularly those in the program who were going to be engaged in computationally intensive research projects, would benefit greatly from understanding the purpose of sustainable software development along with the tools necessary to engage with it.

2.2 Activity overview

In Table 1, we share the structure of our activity. We began with a short lecture to introduce the idea of sustainable software development and provide examples of various ways that facilitators engage with collaborations and developing software. Afterwards, we transitioned to a "Raising Questions" prompt, dividing learners into small groups designed to elicit thoughts and questions about what sustainable software development might look like and emphasizing how it might look different for communities of various sizes. We defined four internal, i.e., unknown to the learners, categories of questions based on the workshop content: issues, roles, code changes, and miscellaneous. As the learners came up with questions, we collected and sorted them into the categories. We then led a discussion for the learners to determine their own names for the categories. In general, the names they determined matched our categorization.

The first portion of the workshop had learners address the following prompt in small groups: "Create a project repository. Experiment with branches and pull requests and think about how they fit within a scientific software development workflow for a student-advisor collaboration." We emphasized beginning with a student-advisor collaboration because that would be authentic to the learners' REU activities and because it generally requires the simplest workflow. During this time, we presented an additional prompt with facilitation to discuss the git tools (branches, merge requests, and issues) needed to enable such a workflow. We ended this portion of the workshop by having the learners form new groups (sometimes referred to as a "jigsaw") and share what their groups thought about with respect to different software communities and the git tools.

The second main portion of the workshop built on the exploration from the first portion. Learners were asked the following prompt in their small groups: "Write a software development workflow document on the repository. Test all aspects of your workflow with examples of your choice." The learners were encouraged to think about larger and

Table 1: Activity Overview. This table outlines the flow of our activity, including time spent on each portion and the accompanying facilitation prompts.

Section	Time	Participant Structure	Prompt given to learners that drives this component
Introduction	10 min	'Mini lecture'	Brief intro to the importance of (collaborative) software development with an emphasis on linking to real world examples in different areas.
Raising questions	20 min total	Small groups (3-4)	Prompt: Broadly think about collaborative software development from small to large projects. Write down questions, concerns, or general topics of interest pertaining to challenges and processes in different collaborative software development environments.
	15 min		Additional facilitation or prompt: State that students should consider questions about small to large communities like those they might contribute to over their summer research program. Facilitators roam the room, take questions as they write them, and sort them into our 4 categories (issues, roles, code changes, misc.). Pin up questions on the board (without category titles yet) as they are raised.
	5 min	Full class discussion	Discuss as a class what we might name each category (besides misc.)
Investigations	60 min	Small groups (3-4)	Gave a brief primer on the git commands needed to carry out Prompt 1 and Prompt 2. Prompt 1: Create a project repository. Experiment with branches and pull requests and think about how they fit within a scientific software development workflow for a student/advisor collaboration.
			Prompt 2: Explore making and managing issues on GitLab and how they relate to branches and pull requests. Consider how using issues is useful in a scientific software development workflow within a moderate size collaborative development group.
			Additional facilitation prompt: Consider what roles and responsibilities developers and scientists have in a large software development community. In what different ways does the community interact with the repository (i.e. branches, pull requests, etc.)? What responsibilities may be assigned to which groups?
	15 min	New small groups (3-4)	Prompt: Share what you learned about how branches, pull requests, and issues fit into a workflow for different scientific software development groups and communities.
	40 min	Original small groups (3-4)	Additional facilitation prompt: Write a software development workflow document on the repository. Test all aspects of your workflow with examples of your choice. Announce that preparation of the culminating assessment task will follow.

Section	Time	Participant Structure	Prompt given to learners that drives this component
Culminating assessment task	30 min = 10 min (prepare) + 5 min (transition) + 15 min (jigsaw)	Jigsaw (3 groups for three facilitators)	Prepare to describe to learners outside of your group your workflow and justify how its design supports a large collaborative software development community. Facilitation prompts: What are the key elements of your workflow and which challenges of (collaborative) software development did you address with it? Did you encounter any problems in executing your workflow? Did you observe/experience anything else you'd like to share?
Synthesis	5 min	'mini-lecture'	Closing remarks including a link back to the motivation.

more complex collaborations and to test their workflows as they developed them. They then engaged in a second jigsaw to describe their workflows to their peers. Finally, we presented a short synthesis lecture where we returned to the questions they raised at the beginning of the workshop and connected them to the key points of a successful workflow.

2.3 Assessment strategy

To assess the learning outcomes of our workshop, we used multiple strategies. We emphasized jigsaws to ensure that all learners were able to form a level of confidence in their knowledge and so facilitators could gauge the learners' progress. Because the learners were engaging with online git repositories throughout the workshop, we were also able to view their explorations through their git repositories as they happened, as well as after the workshop. The main artifact from the workshop was the software development workflow that each group created and tested in the second half of the workshop. We assessed those workflows in the jigsaws and in written form against our rubric for content objectives (see Table 2).

3. Activity development

3.1 Learning outcomes

When version control with git is introduced, it is often presented as a list of very particular commands to be executed from the terminal without much motivation for its usage. To better teach the concepts of sustainable software development, we used an inquiry learning approach to facilitate deeper understanding and make using git more approachable for all learners. Additionally, we used GitLab due to its availability, although GitLab is just one of many hosting sites for version control. We wanted learners to leave our workshop empowered to use any version control tool.

We determined that the main components of a robust software development workflow are issue/bug management, making code changes, and role management. Our rubric (shown in Table 2) shows how we assessed how well those components were incorporated into their workflows. For issues/bug management, learners should ideally include a process to report issues/bugs, guidelines for creating issues to give sufficient detail to fully describe a problem, make a plan for determining responsibility for addressing a given issue, and develop a scheme for prioritizing and fixing the issues. For making code changes,

Table 2: Assessment Rubric. This table details the rubric we used to measure the learners' understanding of the components of our workshop.

Dimensions: Components or "knowledge statements"	M evidence needed to make a judgment is missing	0 evidence that learner has misunderstanding or incomplete understanding	1 evidence that learner has sufficient understanding
Issue/bug-management	No guidelines given for reporting problems	Guidelines to report problems are minimal/incomplete Bugs are only fixed in private branches Not enough information to communicate issues (Such information could be reproducibility for bugs or motivation for feature requests)	There is a process to report issues/bugs Issues fully describe the problem (ideally include minimal working examples / also "full" information is flexible) Someone is responsible for an issue Prioritization
Making code changes	Workflow does not address guidelines for making code changes robustly	Learners make code changes directly on the main branch Process to test code is minimal Merge without approval Code changes are not described in detail	Learners create a workflow that, e.g., includes Making an own branch/fork with a descriptive name Make all changes locally Testing the code Submit a merge request (incl. documentation) Follow up on comments Merge request need approval Merge actually happens
Role management	Roles are not defined and/or assigned	Roles are given but permissions not clearly defined Some project members have too much or too little responsibility for the code All developers have access to the main branch	Roles are clearly defined, e.g., developers, maintainers, users Roles are clearly communicated All aspects of the software development are assigned/linked to roles and there's at least one person per role

learners should ideally include guidelines for giving descriptive names for branches/forks, a process for making changes locally first, a process for testing code throughout development, a process for submitting a merge request (including documentation), a process for following up on any comments, and developing a plan for approving and implementing merges. For role management, learners should clearly define and communicate the roles of the community, have all aspects of the workflow assigned and/or linked to roles, and ensure all roles are filled.

In addition to our main components, we also included two additional dimensions in our rubric: First, for the implementation of a STEM practice — as defined within the PDP (Metevier et al. 2022a, 2022b) — we had learners design a solution within requirements. Their workflows needed to facilitate sustainable software development in a straightforward way. We desired for learners to design a workflow that suited their community, had a plan for each major component, and included reasoning for the choices they made. Second, we added an additional dimension that the design process itself was collaborative — making the process itself more authentic. As the learners developed and tested their workflows, they themselves engaged in an example of sustainable software development and collaboration. Learners needed to work together to determine their final workflow and to include justification for their decisions.

3.2 Content development highlights

When developing our workshop within the PDP, we focused on designing an inclusive workshop that would help learners build their STEM identities. Because REU programs often introduce undergraduates to practicing scientific research, we wanted to create a workshop that would be inclusive to all experience levels. Our workshop design used text files instead of code to avoid prerequisite knowledge of a programming language. We also used the browser version of GitLab rather than command line git to include learners who may not be familiar with using the command line.

Sustainable software development requires collaboration, so we designed our workshop to have learners collaborate with each other while developing their workflows. This provided an opportunity for learners to see the value in sustainable software development as they participated in the workshop.

We began our workshop by introducing the variety of connections with software development we have in our own work to emphasize how sustainable software development applies in practice and connect with our learners. Then, we had the learners engage in a raising questions activity to introduce the central ideas of sustainable software development. During the synthesis portion of the activity, we returned to the questions that were brought up in the raising questions portion and connected them to the concepts they explored. Our goal with this design element was to provide an opportunity for the learners to connect what they learned to their own thoughts and experiences with collaboratively developing a software workflow. Furthermore, emphasizing the value of the learners' questions and their contributions to the learning process provided an opportunity to build ownership of the material (Metevier et al. 2022a, 2022b).

In our design, we included several components with the goal of having our learners build a STEM identity. By implementing periodic jigsaw discussions, we were able to have learners build confidence and independence in the material as the workshop progressed. We were also able to assess their progress throughout the workshop which allowed additional facilitation. With our synthesis lecture, we provided recognition of the work they did and connected their work to real-life examples, both from the facilitators' experiences and the experiences of the learners. Because our workshop was designed to facilitate the use of sustainable software development in their summer projects, we connected the workshop content to potential implementations in their

projects. Our text-based exploration of git also prepared learners to use git for other things beyond code development such as for paper writing, lab notebooks, and documentation.

3.3 Pivot to virtual in 2020 and 2021

In 2020 (and 2021) the COVID19 pandemic prevented in-person REU programs at MSU. Given that all REU projects were conducted remotely, the virtual nature of students' projects made using sustainable software development — especially with centralized, collaborative version control systems — became even more important. Therefore, we adjusted the workshop so that we could deliver it in a virtual format via Zoom (an online video conferencing software). Our main goal for the virtual format was to keep all the essential components we originally designed in place and limit the changes to technical aspects.

In particular, we employed the breakout room capability of Zoom to reflect the original work in small groups. As facilitators, we moved between rooms to listen to conversations and facilitate where necessary, similar to moving between group desks in the in-person format.

For the raising questions component, we employed virtual whiteboards (technically a Google Doc) that allowed all learners to add their questions and ideas simultaneously to a shared space. Again, this component reflected the original collection of questions in the in-person format and allowed us to collect and sort in the background.

A major change pertained to the technical components of the workshop, such as creating a repository, sharing it with other learners, or evaluating/trying the designed workflow. Here, we reused selected submodules of the Software Carpentry Git workshop (Wilson 2006, 2013). These submodules already contained detailed instructions that allowed each learner to progress at their own pace. We leveraged those existing technical instructions and facilitated joint problem-solving and discussions in small groups in breakout rooms. Therefore, we could focus on our content goals around collaborative software development rather than technical aspects.

Finally, the resulting artifacts were the same as for the in-person workshop. We were able to evaluate the outcomes by examining the repositories created by the learners during the workshop.

3.4 Discussion of learner outcomes and artifacts

Our content goals were for learners to understand issues/bug management, how to make code changes, and how to manage roles when developing code within large and small software development communities. Learners with less prior coding experience struggled to envision how to handle bugs and code changes, but all learners were able to grasp the idea of roles and how they could be applied. Interestingly, learners with more prior coding experience seemingly thought more deeply about issues/bug management and making code changes but needed varying degrees of facilitation to begin considering roles within a development community. Learners did a good job developing a workflow but struggled to determine how to test their workflows, although this was likely due to limited time. Some groups were able to test their workflows, but most ran out of time.

We assessed their understanding by applying our rubric to the document each group made to describe their workflow and to their corresponding repositories. We were able to informally assess understanding through a jigsaw discussion where each learner described their group's workflow. Learners were given a score of 1 if they showed sufficient understanding, 0 if they showed incomplete understanding, and M if the content was missing. We did not have any learners where the content was missing, but there were some instances where learners didn't fully address some of the content goals.

This activity was interesting to lead since we taught the workshop twice, and in one class, everyone had prior coding experience while in the other class,

few learners had prior experience. The coding experience of these groups, given the physics versus computational focuses of their respective REU programs, were also opposite of what we had expected before leading the workshop. Based on our assessment, we believe that an additional ~60 minutes would have been helpful to ensure that all groups would be able to explore testing their workflows. Overall, however, our activity worked to get the learners to understand our concepts. We believe our approach of emphasizing the process of sustainable software development instead of the specific commands and jargon used in version control worked well.

The STEM practice goals we incorporated into our activity were to design a solution within requirements and to experience a collaborative design process. This former process is authentic to STEM because we often develop codes or devices that carry out a desired purpose within certain constraints. We assessed the practice with our STEM practice rubric by looking at their repositories and gauging their familiarity with the concepts during the Culminating Assessment Task (CAT) jigsaw. Learners struggled with the idea of determining the requirements for their project, but they did well at realizing that there is more than one solution and were able to develop solutions that fit requirements. When struggles with determining the requirements arose, we facilitated discussion within the groups primarily using the additional facilitation prompt from the Investigations section in Table 1. The prompt asks the learners to consider the ways in which one might interact with the workflow and what their roles might be. We also encouraged them to think about some of the challenges that may arise if there are not sufficient guidelines for a workflow.

Overall, learners worked well with each other to come up with a final solution for their group. In general, the learners were able to work together to form a final workflow document that everyone in their group agreed on. We were able to facilitate this process in part by our instructional design where we emphasized that the design of a software development workflow is inherently collaborative and an authentic practice in a software community. In one case, a group created their own framework (modeled after the US government) and assigned people themed roles. They not only created a set of norms that would work for a software community, but were also creative in their solution.

3.5 Lessons learned

During the development of this workshop, we explored new realms in applying the PDP framework to teach computational concepts. We successfully implemented an inquiry approach and created a successful workshop. In particular, the inclusive design was ideal for our venue since it allowed learners to begin building an identity as participants in a software community, regardless of their prior experience with version control. Because active learning results in better retention of concepts (Hake 1998) it is our hope that our approach can result in a better understanding of how to train scientists in practicing sustainable software development.

4. Conclusion

In developing this workshop through the PDP, we applied inquiry learning and backward design to teach computational concepts and tools. Furthermore, we improved on the way that sustainable software development is introduced to learners. Since sustainable software development can be done effectively with a variety of tools, we emphasized the concepts (issues/bugs, making code changes, and role management) instead of solely presenting the tools (git) to implement these concepts. By having learners interact primarily with the web browser version of GitLab, we facilitated understanding the concepts prior to the learners gaining proficiency in tool usage. After participating in the workshop, learners should be able to apply sustainable software development practices to their own projects, expanding on their knowledge of git if necessary.

The workshop described here was developed as a three-hour workshop. But in principle, this approach could be effectively implemented in a classroom setting as well. The process of working collaboratively in small groups to create a workflow is an authentic experience both in developing software and working with a software community. Some of the learners that participated in the workshop were not directly involved in computationally intensive research projects, so the workshop was less immediately applicable to them. However, the ubiquity of writing code in STEM fields and beyond makes engaging in this workshop a worthwhile professional development opportunity for learners.

Acknowledgements

We thank the MSU Department of Physics & Astronomy, MSU Department of Computational Mathematics, Science, and Engineering, and ISEE who funded our 2019 PDP participation. The PDP was a national program led by the UC Santa Cruz Institute for Scientist & Engineer Educators. The PDP was originally developed by the Center for Adaptive Optics with funding from the National Science Foundation (NSF) (PI: J. Nelson: AST#9876783), and was further developed with funding from the NSF (PI: L. Hunter: AST#0836053, DUE#0816754, DUE#1226140, AST#1347767, AST#1643390, AST#1743117) and University of California, Santa Cruz through funding to ISEE.

References

Better Scientific Software (BSSw). (2022). Retrieved from https://bssw.io/

Carlone, H.B., & Johnson, A. (2007). Understanding the science experiences of successful women of color: Science identity as an analytic lens. *Journal of Research in Science Teaching*. 44, 1187-1218. https://doi.org/10.1002/tea.20237

Guzdial, M. (2010). Why is it so hard to learn to program? In A. Oram & G. Wilson (Eds.), *Making Software: What Really Works, and Why We Believe It* (pp. 111–124). Sebastopol, California: O'Reilly Media, Incorporated.

Guzdial, M. (2013). Exploring hypotheses about media computation. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER'13)*, Association for Computer Machinery, New York, NY, 19–26. https://doi.org/10.1145/2493394.2493397

Hake, R. (1998). Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, 66, 64-74. https://doi.org/10.1119/1.18809

Hazzan, O., Lapidot, T., & Ragonis, N. (2011). *Guide to teaching computer science: An activity-based approach* (First edition). London, England: Springer. https://doi.org/10.1007/978-0-85729-443-2

Metevier, A. J., Hunter, L., Seagroves, S., Kluger-Bell, B., McConnell, N. J., & Palomino, R. (2022). ISEE's inquiry framework. In *ISEE professional development resources for teaching STEM*. UC Santa Cruz: Institute for Scientist & Engineer Educators. https://escholarship.org/uc/item/9q09z7j5

Metevier, A. J., Hunter, L., Seagroves, S., Kluger-Bell, B., Quan, T. K., Barnes, A., McConnell, N. J., & Palomino, R. (2022). ISEE's framework of six elements to guide the design, teaching, and assessment of authentic and inclusive STEM learning experiences. In S. Seagroves, A. Barnes, A. J. Metevier, J. Porter, & L. Hunter (Eds.), *Leaders in effective and inclusive STEM: Twenty years of the Institute for Scientist & Engineer Educators* (pp. 1–22). UC Santa Cruz: Institute for Scientist & Engineer Educators. https://escholarship.org/uc/item/9cx4k9jb

Nangia, U., & Katz, D. S. (2017). Track 1 paper: Surveying the U.S. National Postdoctoral Association regarding software use and training in research (Version 3). *figshare*. https://doi.org/10.6084/m9.figshare.5328442.v3

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? *Communications of the ACM*, 56(8). https://doi.org/10.1145/2492007.2492020

Queiroz, F., Silva, R., Miller, J., Brockhauser, S., & Fangohr, H. (2017). Track 1 paper: Good usability practices in scientific software development (Version 3). *figshare*. https://doi.org/10.6084/m9.figshare.5331814.v3

Sorva, J. (2012). *Visual program simulation in introductory programming education* (Doctoral thesis, Aalto University, Espoo, Finland). Retrieved from https://aaltodoc.aalto.fi/handle/123456789/3534

Wilson, G. (2006). Software carpentry: Getting scientists to write better code by making them more productive. In *Computing in Science & Engineering*, *8*(6), 66-69, Nov.-Dec. 2006. https://doi.org/10.1109/MCSE.2006.122

Wilson, G. (2013). Software carpentry: Lessons learned. arXiv:1307.5448. https://arxiv.org/abs/1307.5448