No Science Software is an Island: Collaborative Software Development Needs in Geosciences

Yolanda Gil
Information Sciences Institute
University of Southern California
gil@isi.edu

Eunyoung Moon
School of Information
University of Texas at Austin
eymoon@utexas.edu

James Howison
School of Information
University of Texas at Austin
jhowison@ischool.utexas.edu

ABSTRACT
Professional software practices increasingly involve software sharing and collaborative development of software. As science becomes an increasingly collaborative enterprise, is there any increasing need for collaborative software practices? We collected data from geoscientists in early career stages with diverse research areas. Although they had varying software development skills, they consistently emphasized the need for improved software sharing and reuse. Moreover they wish to learn more about modern software sharing, open source communities, and collaborative software development practices as they become more interested in various aspects of software stewardship. We briefly examine the current educational resources that early career scientists may have encountered and note that very few address the issues raised by our respondents. Accordingly, we argue that these aspects of work in today's science ought to be incorporated in scientific method and education curricula for scientists. We conclude with preliminary strategies for addressing this.

Author Keywords
Software sharing, software reuse, scientific software.

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces - Interaction styles

General Terms
Human Factors; Design; Measurement.

INTRODUCTION
"No man is an island,
Entire of itself,
Every man is a piece of the continent,
A part of the main…"

– John Donne, Meditation XVII, (1624)

Submitted to WSSSPE 2.0.

Software development is increasingly a collaborative enterprise. In science, software sharing and reuse is crucial for effective and efficient use of research funding and to reach goals of the scientific method, including correctness, transparency and the ability replicate and extend the work of others (Katz et al., 2014).

Existing research has found, however, that scientists developing software have tended to do so in disconnected groups, if not individually (Boisvert and Tang, 2001; Howison and Herbsleb, 2013). In this manner scientists may simply be following incentives to earn individual reputation or at least avoid the overhead of collaboration.

Yet an alternative hypothesis is that scientists are aware of the value of collaboration and wish to work more collaboratively, but are held back by a lack of focus on these topics in their education, particularly in their software education.

The question is important for knowing which policy responses to emphasize: providing improved educational materials for collaboration is substantially easier and more immediately actionable than first seeking to address questions of incentives.

SUBJECTS
In order to investigate this, we approached geoscientists regarding their current software practices and areas where they would want to learn more about software. We approached 40 individuals who had participated in workshops and were early career researchers. We were seeking a population that was more likely to have been exposed to programming, and that might seek to learn more about software through their careers given the trends in science. We received 27 responses.

We solicited a letter from each individual, asking them to express their needs in terms of software skills. Each was asked to include a paragraph about their research interests, followed by one or more paragraphs on their current practices in software and topics that they would like to learn more about. The letters were otherwise not required to be structured; instead they were free form responses. We did not prompt them to respond to specific topics, instead we let them raise the topics that naturally came to mind when thinking about software and the issues they confront in that area.

We analyzed the diversity of the respondents, in terms of their research areas, the positions they hold, and their home institutions. We analyzed their research areas based on the research topics mentioned in the paragraphs about their research interests. The signature included in the letters mentioned the position held by the respondents and their home institution.

We annotated the letters that we received in two different ways: first we assessed the software skills of the respondents, then we assessed the topical areas the participants emphasized. We reasoned that those with substantially greater software skills would emphasize different aspects than those without.

To assess the software skills of the respondents, the letters were analyzed in terms of the language used to describe their current software practices. Based on this analysis, the annotators classified respondents into three broad categories according to their programming skills and familiarity with computer software practices:

- **Non-programmers**: Scientists that have very minimal knowledge about programming and software practices.
- **Developers**: Scientists that develop software, and do some form of software publication and sharing.
- **Advanced developers**: Scientists that have advanced programming skills and have sophisticated knowledge of software sharing and open source practices. Many release their software in open source repositories.

The second annotation concerned the software issues mentioned by the respondents. The letters were analyzed in terms of the language used to describe topics in software development and stewardship that the respondents felt would be beneficial for them to learn about. We created a set of topics mentioned in the letters, taking into account that different language was used to describe a given topic. For example, a letter mentioned "with an open source repository and community-contributed code" and was tallied as the topic of "Building communities around models/codes."

FINDINGS

Diversity of Respondent Population

Figure 1 shows a "wordle diagram"[1] that illustrates the diversity of geosciences topics taken from the short descriptions of research interests in their letters. The topics cover the major areas associated with geosciences research: Earth, Ocean, Atmospheric, and Arctic sciences.

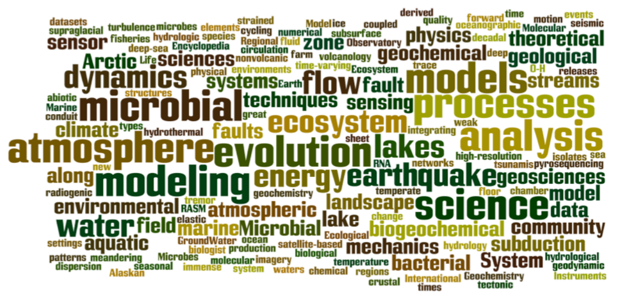

Figure 1. An illustration of the geosciences research topics represented by the respondents.

The respondents hold positions usually associated with early career researchers including assistant professor (9), research assistant professor (3), associate professor (2), post-doc (4), PhD student (4), and other junior positions (5). 13 of them are women.

Their home institutions included academic departments (22), research institutions (3), and government organizations (2). They cover 16 states: AL (1), AK (1), AZ (1), CA (5), CO (1), CT (1), DC (1), HI (1), MA (1), MI (1), NY (2), OH (1), OR (2), SC (2), TX (2), UT (1), and WI (3).

Software Sophistication of the Respondents

The respondents had varying levels of sophistication in terms of writing and using software, using open source software, and contributing to open source software communities. 3 were non-programmers, 18 developed software for their work, and 7 had advanced software development skills. The distribution is shown in Figure 2.

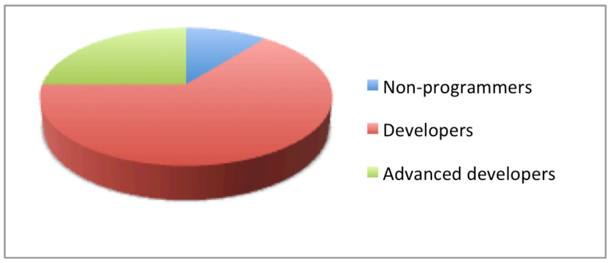

Figure 2. Software skill level of the respondents.

Software Needs

Table 1 summarizes the topics mentioned in the responses referring to needs in software development and open source practices. We grouped the topics into five major themes: 1) preparing software for sharing, 2) describing software, 3) open source software practices, 4) software reuse, and 5) science practices and software. We show the total number of respondents that mentioned a topic, as well as a breakdown based on their software sophistication.

Note that given our approach to data collection, a respondent would mention their 3-5 top-rated topics and might not go on to mention other topics even if they would be important to them. In other words, not mentioning a

[1] http://www.wordle.net

Issues	Mentions	Breakdown		
		NP	DEV	ADV
Preparing Software for Sharing				
Writing software that is easier to integrate	1			1
Making code more portable	2		2	
Documenting software for distribution	**10**		**6**	4
Improving software sharing	**12**	1	**9**	2
Describing Software				
Comparing different modeling codes	3	1	1	1
Linking codes as workflows	2		1	1
Making software accessible to non-programmers	**6**		6	
Open Source Software Practices				
Availability of archives to distribute software	**7**		5	2
Open source software practices	**6**		2	4
Developing communities around models/codes	**8**		7	1
Managing software updates	**8**		5	3
Software Reuse				
Reusing software for data preparation	**7**		5	2
Reusing visualization codes	**3**		3	
Reusing codes from others/ reusing legacy codes	**12**	1	7	4
Science Practices and Software				
Making work reproducible by releasing software used	1			1
Interpreting data using software as provenance	4	2	1	1
Being more efficient at developing software	**11**	1	**10**	
Mechanisms for credit and citation, for justification of effort	**9**		**9**	
Facilitating training of new researchers	**6**		4	2

Table 1. The software topics that were highlighted in the 27 responses, shown in the first column and grouped into 5 major themes. The second column shows the number of respondents who mentioned that topic. The mentions are broken down based on the respondent: non-programmer (NP), developer (DEV), advanced developer (ADV). Darker blue indicates 10 or more mentions, lighter blue 5-9 mentions, and light grey 1-4 mentions.

topic in a letter was not an indication that the respondent did not consider it important.

The results in Table 1 show the annotations of our primary content analysis. To confirm that what we were seeing was not overly influenced by any single person's perspective we had a second researcher read the letters and apply the same categories. The results were not identical but confirmed the overall picture reported in this paper.

The results show that traditional software education concerns are relevant to early career scientists: particularly among non-expert developers there was considerable interest in learning to develop software more efficiently.

Similarly, questions of incentives that may undermine collaborative software development were not absent in these results: mechanisms for citation and credit for the development of software were mentioned often. Respondents mentioned that it is hard to justify the investment required to learn software skills and to develop software given that there are no clear mechanisms to benefit their careers. For early career people, this is particularly important. Since they tend to have more software development skills, when work in larger collaborations they tend to be given software tasks while more senior researchers that lack programming skills might focus more on doing the science.

Nonetheless, it is striking that the majority of the software issues raised are concerned with collaborative software development and sharing, rather than individual software development skills. There was a strong interest in improving software documentation so it can be easily understood by others and reused. Another highly desirable set of skills concerned improving software sharing. Many wanted to share their software, but they did not know how to do that. The development of communities around software codes was another important issue. Many of the respondents have released their software in a software sharing site (e.g., GitHub or similar), and had a community of contributors which in many cases was causing them more work than they would like. They were looking to learn how to manage this contributor community. They were also interested in best practices in managing software updates when there is already a community that has adopted their software.

Another need that was mentioned very often was helping non-programmers to reuse software. Non-programmers included not only colleagues but also students. Students not only have to come up to speed in the particular science they are studying, but also on the software. Having mechanisms that allow others to learn quickly how to use scientific software was considered important.

DISCUSSION

The results show that early career scientists are concerned with incentives and do want to improve their individual programming skills. Yet there was widespread interest in a acquiring skills relating to collaborative software development, including matters of attribution and credit.

We think it reasonable to conclude that while incentives are of concern, early career scientists are not letting uncertainty about those incentives stop them from seeking to share, reuse and collaboratively develop software.

How then might these early career scientists learn about the topics they indicate interest in? When and where in their careers might they encounter this material? While a systematic survey of the educational experiences of this group might shed more direct light on these questions, as a preliminary measure we examined three potential sources: the scientific computing curriculum at a major research university, tutorials at Supercomputing and the curriculum of Software Carpentry, a leading edge software education for early career scientists.

We briefly examined the scientific computing curriculum at a major research university. As expected the emphasis of the scientific computing curriculum is heavily technical, focusing on programming languages and mathematical techniques (e.g., vector optimization, profiling), assistance in using particular hardware, and courses focused on specific scientific domains. There were no course titles that indicated training in collaborative development, participation in open source projects, or issues arising in re-using the code of others.

We then examined the tutorials offered at the Supercomputing conference in 20 2013, and 2012 (excluding the WSSSPE1 workshop at SC 2014). The vast majority of these are education in using specific tools or optimization; one 2012 course mentions "co-design" but none focus on the collaboration issues raised by our early-career respondents.

Finally, we examined the curriculum of Software Carpentry, considered the leading edge program for introducing science students to computing best practices. The primary focus is on individual efficiency and introducing specific tools, but this curriculum definitely addresses collaboration: they teaching collaborative source code management, specifically addresses "Open Science" and the open source licenses and discuss options for hosting code (such as Github and BitBucket). In addition the recommended reading include books on managing open source software projects.

Although our assessment is very preliminary it highlights that there is a very limited focus on issues of collaborative software development, reuse and software community management available within the standard educational experiences of early-career scientists.

CONCLUSION

While incentives for improved software practices are in the minds of early career scientists, there is a clear and surprising interest in improving skills in software reuse and collaboration shown in our study. Thus it seems essential to include units on these topics within computational science education generally. Indeed since software is increasingly how much science is undertaken, we should argue that collaboration and software reuse is an important part of scientific methods and work to encourage its inclusion in mainline scientific education. One way forward would be to work with the Science of Team Science initiative driven by NIH (Falk-Krzesinski et al., 2010) and ensure that software work is included in curriculum initiatives coming out of that community.

REFERENCES

Boisvert, R.F., Tang, P.T.P. (Eds.), 2001. The Architecture of Scientific Software.

Falk-Krzesinski, H.J., Borner, K., Contractor, N., Cummings, J., Fiore, S.M., Hall, K.L., Keyton, J., Spring, B., Stokols, D., Trochim, W., Uzzi, B., 2010. Advancing the Science of Team Science. Clin. Transl. Sci. 3, 263–266. doi:10.1111/j.1752-8062.2010.00223.x

Howison, J., Herbsleb, J.D., 2013. Incentives and integration in scientific software production, in: Proceedings of the ACM Conference on Computer Supported Cooperative Work. San Antonio, TX, pp. 459–470. doi:10.1145/2441776.2441828

Katz, D.S., Choi, S.-C.T., Lapp, H., Maheshwari, K., Löffler, F., Turk, M., Hanwell, M.D., Wilkins-Diehr, N., Hetherington, J., Howison, J., Swenson, S., Allen, G., Elster, A.C., Berriman, G.B., Venters, C.C., 2014. Summary of the First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1). CoRR abs/1404.7414.

Wilson, G., 2014. Software Carpentry: lessons learned. F1000Research. doi:10.12688/f1000research.3-62.v1