Report of the 2017 NSF SI2 PI Workshop

Arlington, VA, 21–22 February 2017

REPORT AUTHORS

Ganesh Gopalakrishnan	University of Utah
Yung-Hsiang Lu	Purdue University
Matthew Knepley	Rice University
Kyle Niemeyer	Oregon State University
Matthew Turk	University of Illinois Urbana–Champaign
Nancy Wilkins-Diehr	University of California, San Diego

Website: https://sites.google.com/view/2017-si2-pi-meeting
https://si2-pi-community.github.io/2017-meeting/

This material is based on work supported by the National Science Foundation under award number ACI-1702722. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Contents

1 Introduction

NSF's vision of a Cyberinfrastructure Framework for 21st Century Science and Engineering identifies advancing new computational infrastructure as a priority for driving innovation in science and engineering. Innovation occurs through advances in computing facilities, scientific instruments, software environments, advanced networks, data storage capabilities, and the critically important human capital and expertise. Software is thus an integral enabler of computation, experiment and theory and a central component of the new computational infrastructure. The Software Infrastructures for Sustained Innovation (SI2) program fosters this vision, and promotes research through several funding instruments. It has built a vibrant community of researchers who hold annual principal investigator (PI) meetings in the form of a workshop.

In this series, the fifth SI2 PI workshop was held at Westin Arlington Gateway on February 21–22, 2017. The website for the 2017 meeting is on the coverpage and this link also leads you to the websites of the 2013–2016 workshops. The organizers for the 2017 event were:

- Ganesh Gopalakrishnan (PI), Professor, School of Computing, University of Utah, http://www.cs.utah.edu/~ganesh
- Yung-Hsiang Lu (co-PI), Associate Professor, Electrical Engineering, Purdue University, https://engineering.purdue.edu/HELPS/Faculty/yunglu.html
- Matthew Knepley (co-PI), Assistant Professor, Department of Computational and Applied Mathematics, Rice University, http://www.caam.rice.edu/~mk51/
- Kyle Niemeyer (co-PI), Assistant Professor, School of Mechanical, Industrial, & Manufacturing Engineering, Oregon State University, https://niemeyer-research-group.github.io
- Matthew J. Turk (co-PI), Assistant Professor, School of Information Sciences/Astronomy, University of Illinois Urbana–Champaign, http://www.astro.illinois.edu/people/mjturk
- Nancy Wilkins-Diehr (Senior Personnel), Associate Director, SDSC, University of California San Diego, http://users.sdsc.edu/~wilkinsn/

We now summarize the planning and execution of the 2017 workshop §2 and the salient outcomes focusing on audience breakout sessions and the exit survey §3. We close with lessons for future instances of this workshop §4.

2 Planning and Execution

We present how the SI2 PI meeting's design was approached and planned by the PIs (§2.1). We then present the highlights of the meeting as it played out, *and as observed and recorded by the PIs* (§2.2).

2.1 Approach and Plan

The organizers got together over Google Hangouts about once a month initially, and approaching once a week as the workshop drew closer. A proposal to NSF was written around September, and finalized and submitted around the middle of October 2016. A contract with Conferences and Events of the University of Utah was signed around this time, with Meghan Webb serving as the coordinator. A contract with KnowInnovation was also signed around then, with Tim Dunne and Costa Michailidis serving as coordinators. A room-block was requested of Westin Arlington Gateway and the hotel contract was finalized. The requisite security deposit of $2,000 was kindly waived by Westin, thanks to the indulgence of Alessandro Sifuentes.

From past SI2 attendee feedback, it became clear that while the attendees like the atmosphere of an "unconference" (loosely formatted and organic in terms of the exact agenda), they also liked to have some direction overall. To accommodate this point of view, we designed our workshop by having more structured meetings on the Day-1, but evolving more to an "unconference" format on Day-2. Reproducibility was chosen as a prominent theme to carry through the program.

A significant organizational headache in past years was the provision of upload sites for audience posters. Taking this responsibility up, in addition to all the other responsibilities of running a workshop, can be quite overwhelming. This year, we decided to rely on Figshare as a "one-stop shop" for poster uploads as well as the speaker slides. Figshare agreed to provide a meeting landing page, accessible to all via `https://nsf-si2-pi.figshare.com`, at no charge. PIs attending the meeting were instructed to upload posters themselves, and tag the submission with `NSF-SI2-2017`, which Figshare then used to collect all submissions for the portal.

Key to any successful workshop are the keynotes, and this time we were lucky to be able to field four invited keynotes on Day-1 and an invited keynote on Day-2.
The keynotes on Day-1 were by

- Lorena Barba of the George Washington University ("Title: How to run a lab for reproducible research"),
- Jed Brown of the University of Colorado at Boulder ("Title: Community building through software design"),
- Eric Klavins of the University of Washington, Seattle ("Title: The Aquarium Laboratory Operating System"), and
- Tim Menzies of NC State University (Lunchtime talk, "Title: Understanding software: recent lessons from empirical SE").

The keynote on Day-2 were by

- Tracey Teal of Michigan State University ("Title: Democratizing and supporting software development: a pathway to sustainable software")

In addition, the following presentations were made by NSF program offiers:

- Day-1 was marked by a presentation by Rajiv Ramnath (Program Director, ACI)

and Irene Qualters (Division Director of ACI), describing their respective programs.
- Day-2 was also marked by presentations from Steven Konsek (Program Director, I-Corps), and Jim Kurose (Assistant Director of CISE) describing their respective programs.

2.2 How the Plan Played Out

An overview assessment by the PIs and the organizing crew (KnowInnovation and Conferences and Events) is provided in §2.2.1. We also provide the reader about the themes covered in various keynotes, in §2.2.2.

2.2.1 Overview Assessment by the PIs

The PIs felt that overall the meeting worked according to plans, and that it was an overall positive experience for the participants. Clearly, some of the goals were missed and the unexpected did crop up; they are duly noted in this section at a high level, as well as more objectively in §4.

Audience Strength and Room Capacity: The number of registrants took everyone by surprise (around 140 registering). This created some significant concerns regarding how we were going to allocate seating in our primary ballroom and also arrange for the breakouts. Fortunately, with the experienced KnowInnovation crew (Tim Dunne and Costa Michailides) and University of Utah Conferences and Events lead (Meghan Webb) working together, these issues were smoothly resolved. In fact, in the lead-up to the SI2 PI meeting, Tim, Costa, and Meghan had figured out many of the logistical details and had planned for contingencies. This chemistry was crucial for the smooth experience delivered over nearly two days.

These issues are likely to be different for 2018 and the following years, as (1) NSF is relocating to Alexandria, thus potentially changing the hotel that future organizers might work with. However, given the working relationships with Westin Arlington Gateway, the 2018 organizers might like to continue with them. (2) SI2 is growing in terms of the number of actively funded projects, thus requiring some more planning on whom to invite and when the size of the audience begins to detract from the quality and intimacy of everyone's experience. (3) NSF likes to be more inclusive, allowing more outside attendees (besides the SI2 funded PIs) to benefit from this (rather resource-intensive) PI workshop. All these points are elaborated in §4.

Keynotes: The PIs took to heart the comments from last year where people wanted to hear about topics of general relevance to the software community. Overall, the keynotes were very well-received. The keynote speakers gave uniformly excellent talks, and thus

inspired the audience members to engage well in breakouts. The time allotment also seemed very good (25 min for the talks with a 10 min Q/A period at the end).

Keynote Topic Seeding and Chair Assignment: The PIs worked with KnowInnovation and seeded breakout topics ahead of time into the Google docs that were designated for the various breakouts. The keynote speakers were also informed of these topics. During Day-1, it appeared as if these seeded topics were pursued almost unmodified. During Day-2, the format was a bit more democratic where proposals for topics were solicited live.

The assignments of chairs for the different sections worked well. There was little problem managing the time. Again, KnowInnovation serving as MCs really played out well (as also observed by the NSF Program Officers). Tim Dunne and Costa Michailides are becoming very aware of our community and thus are able to say the right things at the right time, lead the audience well, and also coordinating with the PIs when things change.

NSF Program Officer Participation: All the NSF Program Officers contributed to the workshop by articulating the directions of ACI and of CISE. It was good to be reminded of the importance of what the SI2 PIs are contributing to. Arranging breakfast on Day-2 with selected (junior) PIs and Jim Kurose (Assistant Director of CISE) was a good overall experience for everyone—something worth repeating.

Poster Session: The poster session continues to be problematic. This year, we had attempted to highlight posters by having the PIs affix "dots" (stickers) informing the audience of the career stage of the presenter (early career etc) and the funding stage of the project (early, mid, ending, etc). We also had attempted to force mixing through "scavenger hunts" and similar activities. Unfortunately, the audience feedback (even the PIs' own experience) was that the poster session requires more work.

2.2.2 Keynotes, and Their Gist

Prof. Lorena Barba [1] echoed the theme of reproducibility, this time with her CFD codes in the Python environment. Using Jupyter notebooks, students can follow along with the derivations, supplemented by informative graphics and animations. The open development process makes reproduction by other groups immediate, and clarifies implementation issues that are often hidden in publication.

Prof. Jed Brown [2] from CU Boulder showed how software design can be used to make projects more collaborative and open to community contributions. A central theme was that plug-in architectures internalize open world assumptions, which allow smooth integration of user contributions, removing the counter-productive distinction between developers and users. Many examples of potential pitfalls coming from real-world library development illustrated this point. The aforementioned changes in software architecture must be paired

with changes in the social organization of the project to empower new contributors. Development should be conducted in an open forum, such as a hosting site, and design decision should be discussed openly and respectfully. Not only can this enlarge the community of users/contributors; it avoids the "technical debt" of integrating forks upstream. Finally, an exhortation was given to properly cite software libraries and applications used in the course of research, and an excellent bibliography detailing these arguments was given.

Prof. Eric Klavins [3] of the University of Washington introduced the audience to his Aquarium framework for the formal representation of synthetic biology work-flows. The leverage from abstraction, combined with auto-generated easy-to-follow graphical work-flows, enabled reproducibility in experiments by even lab technicians who have received a very modest amount of training. This level of reproducibility had previously been out of reach, even for very experienced lab personnel at top research institutions. The lab protocols automated through Aquarium thus offer a pathway toward transferable lab protocols and formal codification of experimental procedures.

Prof. Tim Menzies [4] gave a lunch-time keynote where he narrated lessons from empirical software engineering that could be translated into practice by SI2 PIs. The focal theme was quality assurance of software. The key message was that "data makes all the difference." One striking example was test case prioritization where the quality of test suites (with respect to their propensity to trigger failures) went up from about 25% to nearly 75% after adopting data-driven methods.

Prof. Tracey Teal [5] presented the point of view of Software *Not* as a Service. The key observation made was: "In a world of almost infinite data, it is code and software that turns data into information and knowledge." Thus, people in the academia cannot outsource software development. The requisite cultural shift demands that we (1) build local talent, (2) support development, and (3) collaborate.

3 Outcomes: Attendee Breakouts, Exit Survey

We summarize our primary outcomes under two headings: Summaries of attendee breakout sessions (§3.1) and summaries of our exit survey (§3.2). The detailed notes from breakout sessions (transferred from Google Docs) are available in the Appendix.

3.1 Summary of Attendee Breakout Sessions

The 2017 SI2 PI meeting featured three breakout sessions. The first breakout was on reproducibility. The second breakout covered many participant-proposed topics. The third breakout was on education and training for sustainable software development. We summarize below the gist of various breakouts (for details, please see a detailed Google doc maintained at https://drive.google.com/drive/folders/0B8RnYdT2MFgfcFNYSE1BUUtvVGc.

Reproducibility Breakout: This breakout was triggered by the invited talks of Lorena Barba, Jed Brown, and Eric Klavins.

Breakout groups under this theme discussed the definition of reproducibility: Is it procedural or procedural with data? In some fields, the sources of data are intrinsically transient (e.g., biomedical samples) and cannot be preserved. In terms of software, executing software also involves compilers, libraries, operating systems, and hardware. Moreover, some solutions have built-in non-deterministic steps (such as using random numbers). Thus, reproducibility needs to be clearly defined. The benefits of reproducibility needs to be described, for example, as a requirement for publications or a way to publicize projects. It may be necessary for professional societies (such as ACM or IEEE) to create standard procedures. Funding agencies, such as NSF, may need to provide templates to help investigators develop reproducible discoveries.

One breakout group came to the conclusion that researchers who feel strongly about reproducibility should lead by example, and "market forces" would pressure other PIs to follow their collaborators/competitors. An analogy was made to the NSF Data Management Plan, which is required but without explicit requirements. In addition, the benefits of reproducibility and reproducible research need to be made more clear and obvious—simply forcing people (or trying to) will not succeed.

Attendee-Proposed Breakout Topics: Interest in these breakout topics were expressed by the attendees while they registered (such topics were solicited by our registration page). The following topics were discussed:

- **Software metadata for discovery, credit, and preservation:** This breakout discussed creating metadata so that software can be discovered more easily; such metadata may differ (and be more extensive than) that needed for citation. There is no standard way describing software and it is difficult find the right (open-source) software. As a result, researchers may have to create software solving the same problems. This can leads to significant wastes of efforts: the same problems are solved multiple times and each solution is isolated with only a few users. There are several proposed methods for citing software. Future SI2 solicitations may encourage adoption of a specific citation standard so that all projects supported by SI2 can be cited in a uniform way.

- **Sustainability of software projects after initial funding ends:** Multiple breakouts discussed this topic due to high levels of interest. One aspect included how/whether sustainability of a new project is achievable after three-to-five years of support. One possible solution involves developing a new funding model in which NSF awardees may pay other research teams for using and maintaining software. NSF awards typically support new discoveries and rarely support the software tools that lead to or facilitate the discoveries. One possible funding model is to encourage or

require proposal budgets to include items that directly support the development or maintenance of software. Even though such items are allowed now, it is not common for PIs to include such items. Also, many software projects have no established mechanisms receiving financial supports from other research awards.

- **Extensibility in numerical software:** This breakout discussed (and defined) extensibility, which is related to the ease of adding new, major features to software not supported by the original design. Portability is a kind of extensibility, and maintainability is related—the group argued that extensibility is important for software sustainability. Domain specific languages (DSLs) were discussed as one route to extensibility. In addition, some traits common to extensible software include: data abstraction, modularity, factorizable design, and optimal design of class interfaces.

- **Education for software development and release:** This breakout discussed broadening the software development education process to include students, researchers, and faculty outside computer science/engineering disciplines. Instead of relying on traditional semester-long courses, the community should create short courses (such as one week) and training modules. The courses may be online or in-person (such as summer schools). The community may create certification systems so that students, researchers, faculty can obtain certificates of competence in software engineering. Such certificates may be parts of professional development.

- **Developing cloud-based solutions to facilitate software development and data storage:** This breakout conclused that NSF should encourage the creation and development of cloud-based solutions, such as `github.com` for version control and `travis-ci.org` for testing and integration. These well-designed tools can greatly reduce researchers' efforts in maintaining software. Furthermore, proposals should include data and software storage costs in budgets, as an often-ignored cost of research.

- **Growing the user community:** Users and developers are essential for the long-term health of software projects. Most students, researchers, and faculty, however, have little or no knowledge about how to create and cultivate communities of users and developers. It is advised that future SI2 PI meetings can include presentations and discussions about how to create, cultivate, and improve such communities.

Education on Software Development Breakout: This breakout was triggered by the invited talk of Tracey Teal, co-founder and Executive Director of Data Carpentry. This breakout session covered many topics related to education, including how to create guidelines for contributors of open-source software, training new developers, identifying domain-specific knowledge and skills needed for software development, and career paths for software developers in research projects.

One session began as a breakout discussing how training and practices should be required to be described in an NSF proposal, and ended with the organizers inviting a response to NSF CI 2030 from the entire SI2 community. Those in the session observed that software used in research comes from many sources—e.g., commercial packages, independently developed scripts and programs, software associated with instruments. It is important that research teams understand how to use software properly so that research integrity, reproducibility and trust in findings can all be maintained. The group suggested that wording be added to the NSF Grant Proposal Guide requiring a software plan (which would include training) in addition to a data management. The plan would include descriptions of the use and validation of software as well as the training of project participants in proper use.

Those in a separate breakout discussed the design of on demand training for a user community. One PI hosts five-day "user and developer" workshops and pays for every graduate student attendee's costs (from the SI2 award). The workshops are a mix of talks, future topics, and hands-on work. Hackathons (without the talks) have also helped build community very well. This constitutes almost one-third of the annual development of the package. One group uses a GitHub repository with issues, tagging them with various categories so that attendees can identify what they want to work on. Creating small tasks and showing progress can motivate contributors. Sharing the planned feature list and engaging with those suggesting additional features on their prioritization is also a good way to build priority. There was also great enthusiasm in using Git to teach classes. The principles can be taught easily and then Git can be used, both in software development and for the authorship of websites and papers.

Others spend no money on travel, but build community by interacting with the many people who download the code (and find errors). They find such benefit to creating and maintaining the community, "keep kissing the frogs", make small releases, and the community will grow. Jupyter notebooks were discussed as a way to turn users of codes into developers. One attendee observes that the webinars with the most repeat views are those that demonstrate how to do something—changing parameters and seeing what happens. This sort of thing is rarely shown in classes. The more theoretical lectures have far fewer viewers.

Finally, one group discussed the important topic of career paths for people who are essential but don't want to become academic PIs. Career paths are essential to slow the talent drain as industry snaps up top people. Sustainability in a grant-funded environment in general is challenging. Some find that independent research units and quasi academic, national lab scenarios can work as well as redirection of some facilities and administrative (F&A) income at universities to research software staff. For the latter to happen, universities must be convinced that research computing, including software development, is a core facility akin to libraries. Some suggest that NSF could guide universities in this regard.

Job security could also be tied to degree activities, for example building degree programs in scientific computing where research software programmers teach as well. Alumni from

these programs could be a source of donations. Others feel industry could help fund centers if students then went to work for those industries.

The group observed that the current configuration is not sustainable for career path development.

3.2 Exit Survey Highlights

As in past workshops, at the close of the meeting we sent out an exit survey for attendees to give feedback on the workshop organization, structure, keynotes, etc. Unlike in past years, this survey was administered via Google Forms, and also sent out prior to the close of the workshop—dedicated time was included in the schedule for attendees to begin completing it. The survey questions included:

1. Please describe your role/identity in your own words (e.g., computer scientist, software engineer, physicist)
2. Did you get what you wanted out of this workshop? Why or why not?
3. Comments or suggestions for improvements for the 2018 SI2 PI workshop organizers?
4. How did the KnowInnovation-facilitated, participant-driven style of this workshop work for you?
5. How many potential collaborators did you meet?
6. Give one scientific advance that would not have been possible without your software. What about any software?
7. What questions, issues, or problems did you have coming into the conference that you wanted to explore?
8. How many new questions emerged along the way?
9. Would you like to highlight any particularly important questions, issues, or problems?
10. What questions, issues, or problems did you find a good solution for?
11. Would you like to highlight any particularly good solutions you found?
12. Would you like to highlight any particularly important unanswered questions?
13. How many people did you meet that you will touch base with again in the next year?

76 attendees responded to the exit survey, which represents a bit over 50% of the number of registrants. Although this represents a substantial increase over the post-workshop survey response rate from 2016 (only five out of 98 attendees responded), we had hoped to achieve a higher response rate by providing time at the end of the workshop to begin filling out the survey. However, many attendees had already left at this point.

Figure 1 shows the roles self-identified by survey respondents; nearly one-third identified themselves as computer scientists. (Note that some respondents identified themselves with multiple roles.)

In terms of the feedback given, all but six respondents indicated they got what they wanted out of the workshop, at varying levels. Frequent positive feedback focused on productive discussions, high-quality invited talks, networking with people both working in scientific software and potential collaborators, and meeting NSF officers. Negative feed-

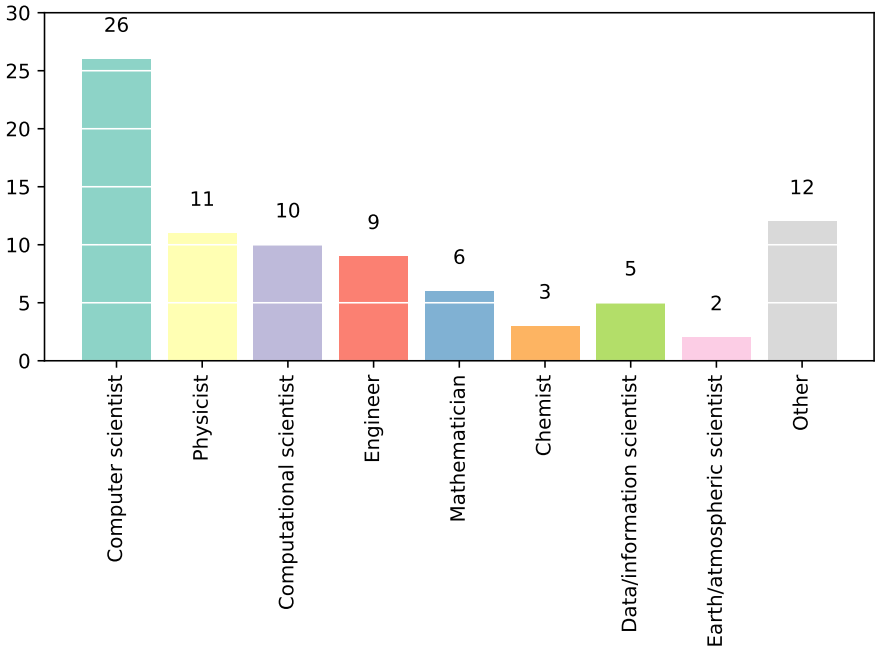

Figure 1: Self-identified roles of exit survey respondents. "Other" includes, e.g., PI, professor, atmospheric/earth/bioinformatics scientist, software developer.

back, on the other hand, mentioned too few people in the respondent's domain, with one comment complaining about a lack of "substance".

Common themes of the suggestions for the next workshop include:

- Schedule: the schedule should be finalized earlier, and include more time for breaks/ have a less-intense first day. In addition, the first day could start later to better accommodate those coming from the West Coast.
- Venue: the venue is not convenient for everyone, and the room barely fits all attendees. In addition, multiple respondents pointed out the lack of protein at breakfast the first day.
- Talk/breakout balance: many respondents commented on the improved quality of invited talks over previous years, and would prefer more time dedicated to those over the breakouts, which some felt included topics of less-than-wide interest.
- Improved website: many commented on the poor quality and lack of content on the meeting website[1], and specifically requested access to an attendee list prior to the

[1] Author note: an improved website was built **after** the meeting, which can hopefully serve as a template

workshop.

- Workshop outcomes: identify clear desired outcomes for the workshop, beyond collaboration and networking.
- Facilitators: continue working with workshop facilitators like KnowInnovation; all but six survey respondents gave positive feedback on their facilitation and the general participant-driven structure of the workshop. Some felt they should take a greater role preventing people from dominating discussions.

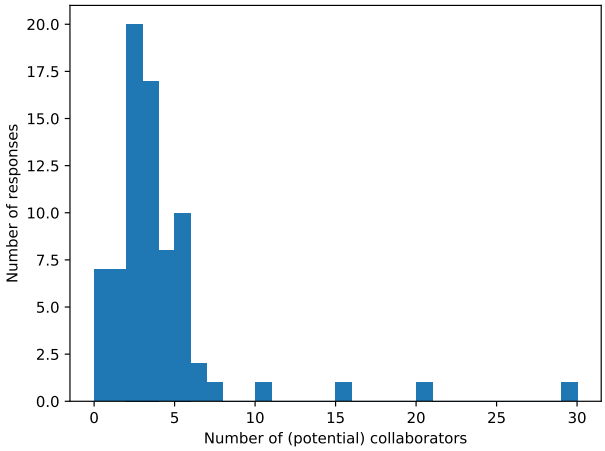

Figure 2: Self-reported numbers of potential collaborators met.

Figure 2 shows the self-reported numbers of (potential) collaborators met by the survey respondents, with an average of 3.6 people met. These results indicate that the majority of respondents made new connections and met multiple possible collaborators, solidifying this as one major benefit of the workshop.

4 Lessons for Future Meetings

The committee, with encouragement from NSF, expanded the meeting beyond the SI2 Program to include other software efforts, such as PIs from EAGER, RAPID, and VOSS awards and the Molecular Sciences Software Institute. The inclusion of outside researchers provided valuable input for the breakout sessions and general discussion, and did not seem to detract from the focus of the meeting.

for future workshops

The committee greatly benefited from the expertise of members who had organized the prior PI meeting. It is highly recommended that some members carry over for each round. In addition, the University of Utah staff were very valuable in the organizational process. It is recommended that the lead PI have some institutional support for meeting organization. We also benefited from the expertise at KnowInnovation, and would recommend this service again. However, the management of process information, as well as the meeting website and registration, could be improved. Currently there is no central repository for this information. Building on the Google Pages site from this meeting, a comprehensive management solution should be implemented in the next round[2]. Moreover, it is important to get an early start on planning. This means that the meeting committee must be selected very soon after the prior meeting.

The selection of speakers was greatly simplified by narrowing the focus of the invited talks. Speakers were chosen to highlight reproducible science and community development. It would likely be advantageous for the next committee to similarly define a theme for the meeting. The committee discussed the balance between speakers and breakout sessions, and opted for a few, main speakers and more time devoted to community discussion. The breakout time was very productive for this meeting, and contributed many new insights. Future meetings might benefit from further thinking on better ways to synthesize and report the findings of breakout sessions; for example, facilitators can focus the breakout groups on outcomes or outputs rather than just discussion, and the report-backs to the larger group should include a (documented) list of takeaways, conclusions, or action items.

One of the biggest challenges facing the 2017 organizers was the increased size of the SI2 community, and the resulting increase in number of attendees for the workshop

A few options, which can be adopted separately or in combination, include:

- Charge a nominal registration fee, for example $100 until an early cutoff date and with an associated poster upload. Then, the registration fee could increase to $200 for late registration or if no poster is uploaded. This mechanism could also be used to allow non-SI2 PIs to attend the workshop and learn about the program (although this does not help the size problem).

  This option does present a few challenges: SI2 PIs did not budget funds to cover this registration fee in their original proposal, and per the proposal and program guidelines PIs are required to attend yearly PI meetings. The latter requirement may need to be relaxed.

- Limit the attendees to one representative per project, including collaborative projects across multiple institutions.

- Set a hard registration deadline, after which attendance will not be permitted. Again, this would be challenging to implement currently based on the required attendance

[2] Following the meeting, an organization was created on GitHub, SI2-PI-community, and a more-complete website via GitHub Pages: https://si2-pi-community.github.io/2017-meeting/

rule. However, in addition to possibly limiting the numbers of attendees, it would also ensure that the workshop size is fixed earlier.

- Find a larger, less expensive venue. The new NSF location may have a better venue nearby, or a less-expensive locale could be selected (if NSF program officers are amenable—one frequently mentioned benefit in the attendee feedback is the ability to talk with multiple NSF representatives).

Acknowledgements

This material is based upon work supported by the National Science Foundation under grant ACI-1702722.

References

[1] Lorena A. Barba. *How to run a lab for reproducible research*. Talk given at 2017 NSF SI2 PI Workshop. Arlington, VA, Feb. 2017. DOI: 10.6084/m9.figshare.4676170.v1.

[2] Jed Brown. *Community building through software design*. Talk given at 2017 NSF SI2 PI Workshop. Arlington, VA, Feb. 2017. DOI: 10.6084/m9.figshare.4676218.v1.

[3] Eric Klavins. *Aquarium: Toward Reproducible Molecular Biology*. Talk given at 2017 NSF SI2 PI Workshop. Arlington, VA, Feb. 2017. DOI: 10.6084/m9.figshare.4684156.v1.

[4] TIm Menzies. *Understanding Software: Recent Lessons from Empirical Software Engineering*. Talk given at 2017 NSF SI2 PI Workshop. Arlington, VA, Feb. 2017. DOI: 10.6084/m9.figshare.4680961.v1.

[5] Tracy Teal. *Democratizing and supporting software development: a pathway to sustainable software*. Talk given at 2017 NSF SI2 PI Workshop. Arlington, VA, Feb. 2017. DOI: 10.6084/m9.figshare.4681288.v1.

A Registrant list

Name	Organization	NSF award number	Award type
Nancy Wilkins-Diehr	San Diego Supercomputer Center	ACI-1547611	SI2
T. Daniel Crawford	Virginia Tech	ACI-1547580	SI2
Amarda Shehu	George Mason University	1440581	SI2
Dhabaleswar K Panda	The Ohio State University	1450440	SI2
Daniel S. Katz	University of Illinois Urbana-Champaign	1550588	SI2
Paul Navratil	Texas Advanced Computing Center	1339863	SI2
David Schloen	University of Chicago	1450455	SI2
Steve Sawyer	Syracuse University	OCI-1221945	VOSS
Lucas A. Wilson	Texas Advanced Computing Center	1550601	SI2
Rajkumar Kettimuthu	University of Chicago	ACI-1339798	SI2
Frank Timmes	Arizona State University	ACI-1339600	SI2
Damian Dechev	Univerity of Central Florida	1440530	SI2
Dmitry Pekurovsky	UC San Diego/SDSC	ACI-1339884	SI2
Carl Boettiger	UC Berkeley	1549758	EAGER
Emanuel Gull	University of Michigan	1606348	CI REUSE
Alberto Passalacqua	Iowa State University	ACI 1440443	SI2
Janos Sallai	Vanderbilt University	1535150	SI2
Sohrab Ismail-Beigi	Yale University	ACI-1339804	SI2
Ganesh Gopalakrishnan	University of Utah	ACI 1535032	SI2
Matthew Knepley	Rice University	1450339	SI2
Kyle Niemeyer	Oregon State University	1535065	SI2
Mark S. Ghiorso	OFM Research	ACI-1550482	SI2
Mark Gordon	Iowa State University	1450217	SI2
Manish Parashar	Rutgers University	1441376	EAGER
Alex Travesset	Iowa State University	1606336	CMMT
Yung-Hsiang Lu	Purdue University	1535108	SI2
Peter Elmer	Princeton University	1558216	SI2
Michael D Sokoloff	University of Cincinnati	1450319	SI2
David Tarboton	Utah State University	1148453, 1148090	SI2
Shantenu Jha	Rutgers	1440677, 1265788	SI2
Richard C Brower	Boston University	1606994	
James Benedict	University of Miami	AGS-1547910	RAPID
Rajiv Ramnath	National Science Foundation		
Volker Blum	Duke University	1450280	SI2
David Hudak	Ohio Supercomputer Center	SI2-SSE-1534949	SI2
David Irwin	University of Massachusetts Amherst	1339839	SI2
Mike Dixon	National Center for Atmospheric Research	1550597	SI2
B.S. Manjunath	UCSB	1650972	EAGER
Stan Ahalt	RENCI/UNC Chapel Hill	1541450	SI2
Ian Foster	The University of Chicago	1148484	SI2
Bruce Berriman	Caltech/IPAC	ACI 1642453	SI2
Ras Bodik	University of Washington	ACT1535191	SI2

Sol Greenspan	NSF		
Larry Frank	University of California San Diego	ACI-1440412	SI2
Inanc Senocak	Boise State University	1440638	SI2
Robert van de Geijn	The University of Texas at Austin	ACI-1550493	SI2
Maggie Myers	The University of Texas at Austin	ACI-1550493	SI2
Philip Maechling	Southern California Earthquake Center	ACI-1450451	SI2
Mark Shephard	Rensselaer Polytechnic Institute	1533581	SI2
Ilya Zaslavsky	UCSD/SDSC	ACI-1443082	EAGER
Namu Patel	Northwestern University	1450374	SI2
Larry Frank	University of California San Diego	ACI-1440412	SI2
Jerry Bernholc	NC State University	ACI-1339844	SI2
Sameer Shende	University of Oregon	ACI-1450471	SI2
George K. Thiruvathukal	Loyola University Chicago	1445347	SI2, EAGER
Shaowen Wang	University of Illinois at Urbana-Champaign	1047916	SI2
Peter Cummings	Vanderbilt University	ACI-1047828	SI2
Coray Colina	University of Florida	1613155	SI2
Joannes J Westerink	University of Notre Dame	ACI-1339738	SI2
Gagan Agrawal	Ohio State University	1339757	SI2
Christopher Iacovella	Vanderbilt University	1535150	SI2
Jan Verschelde	University of Illinois at Chicago	1440534	SI2
Rion Dooley	Texas Advanced Computing Center	1450459	SI2
Ian Anderson	Utah State University	ACI-1642388	SI2
Yifei Mo	University of Maryland-College Park	ACI-1550404	SI2
Charlotte Lee	University of Washington	ACI-1302272	VOSS
Thomas Cheatham	University of Utah	ACI-1521728	RAPID
Laxmikant (Sanjay) Kale	University of Illinois	ACI-1339715	SI2
Michael Norman	UC San Diego	1440709	SI2
Glenn J Martyna	IBM Research		
Miron Livny	University of Wisconsin-Madison	1148515	SI2
Matthew Newville	University of Chicago	1450468	SI2
Chris Rapier	Pittsburgh Supercomputing Center	1534836	SI2
Katy Borner	Indiana University	1566393	EAGER
Wolfgang Bangerth	Colorado State University	OCI-1148116	SI2
Greg Newman	Colorado State University	1550463	SI2, AISL
Jindal Shah	Oklahoma State University	ACI 1339785	SI2
Christopher Roland	North Carolina State University	1534941	SI2
Tim Menzies	North Carolina State University	(speaker)	
Greg Tucker	University of Colorado, Boulder/CIRES	ACI-1450409	SI2
Jason Leigh	University of Hawaii at Manoa	1441963	SI2
Katherine Lawrence	University of Michigan	ACI-1547611	SI2
Anthony Danalis	University of Tennessee	1450429	SI2
Nancy Ide	Vassar College	1147944	SI2
Dani Brake	University of Notre Dame	ACI-1440583	SI2
Jackson DeBuhr	Indiana University	1440396	SI2
Elbridge Gerry Puckett	University of California	ACI 1440811	SI2
Perry de Valpine	University of California, Berkeley	ACI-1550488	SI2

Jing Yang	UNCC	1535081, 1535031	SI2
Youssef Marzouk	MIT	ACI-1550487	SI2
Derek C. Richardson	University of Maryland	ACI1550417	SI2
Thomas Quinn	University of Washington	1550234	SI2
Frank Löffler	Louisiana State University	1550551	SI2
Carlos Maltzahn	University of California, Santa Cruz	1450488	SI2
Randy Heiland	CACR, Indiana University	ACI-1547272	CICI
Shantenu Jha	Rutgers	1440677	SI2
Dane Morgan	University of Wisconsin - Madison	1148011	SI2
Ashish Gehani	SRI	1440800	SI2
Paul T. Bauman	University at Buffalo	1642388	SI2
Barbara Lerner	Mount Holyoke College	1450277	SI2
Jose Fortes	University of Florida	1535086	SI2
Barton Miller	University of Wisconsin-Madison	ACI-1449918	SI2
Umberto Villa	The University of Texas at Austin	1550593	SI2
Xiyin Wang	University of Georgia	1339727	SI2
Sandra Gesing	University of Notre Dame	1547611	SI2
Renato Figueiredo	University of Florida	1339737	SI2
Michael Bell	Colorado State University	ACI-1661663	SI2
Daisuke Kihara	Purdue University	1614777	Venture Fund
Roy Stogner	University of Texas at Austin	1642388	SI2
Anthony Aufdenkampe	LimnoTech & Stroud Water Research Center	1339834	SI2
David Anderson	UC Berkeley	ACI-1550601	SI2
Matthew Turk	University of Illinois	1535651	SI2
Sicong Liu	Arizona State University	1339835	SI2
Andreas Stathopoulos	College of William and Mary	1440700	SSE
Mark Neubauer	University of Illinois at Urbana-Champaign	ACI-1558233	S2I2 concept.
Tim Dunne	KnowInnovation		
Costa Michailidis	KnowInnovation		
Abhijit Majumder	Wayne State University	1550300	SI2
Meghan Webb	University of Utah		
Alejandro	Strachan	1440727	SI2
Reuben Budiardja	University of Tennessee	1535130	SI2
Marlon Pierce	Indiana University	1339774	SI2
Douglas Thain	University of Notre Dame	1642409	SI2
Paul Butler	University of Tennessee-Knoxville/NIST	CHE 1265821	SI2
Philip A. Wilsey	University of Cincinnati	ACI-1440420	SI2
Warren B. Mori	UCLA	ACI-1339893	SI2
Wolfgang Losert	University of Maryland	1550554	SI2
Cameron Smith	RPI	1533581	SI2
Lorena A. Barba	George Washington University	(speaker)	
A. Selcuk Uluagac	Florida International University	1339781	SI2
Alison Marsden	Stanford University	1562450	SI2
Gene Cooperman	Northeastern University	ACI-1440788	SI2
Ed Valeyev	Virginia Tech	1550456	SI2
Abani K Patra	University at Buffalo	1339765	SI2

Mark Shephard	Rensselaer Polytechnic Institute	1533581	SI2
Xiyin Wang	University of Georgia	1339727	SI2
Frank Löffler	Louisiana State University	1550551	SI2
Mercè Crosas	Harvard University	1448123	EAGER
Michael Hucka	California Institute of Technology	1533792	EAGER
Upulee Kanewala	Montana State University	1656877	SI2, SHF
Marco Bernardi	California Institute of Technology	1642443	SI2
Tracy Teal	Data Carpentry	(speaker)	
Chad Hanna	Penn State	ACI-1642391	SSE
Jed Brown	CU Boulder	(speaker)	
Haesun Park	Georgia Institute of Technology	ACI-1642410	SI2
P. Bryan Heidorn	University of Arizona	1642446	SI2
Joannes Westerink	Notre Dame	ACI-1339738	SI2
William Miller	NSF ACI		
James A. Shackleford	Drexel University	1642380	SI2
Andrew Miner	Iowa State University	1642397	SI2
Geof Sawaya	University of Utah		
Eric Klavins	University of Washington	1317653	Expeditions
Jakub Kurzak	University of Tennessee	1642441	SI2
Anton Van der Ven	University of California Santa Barbara	1642433	SI2
Edward Valeev	Virginia Tech	1450262	SI2

B Detailed notes from breakout discussions: reproducibility

B.1 What is the "Goldilocks" level (just right) of reproducibility for each area?

Moderator: Lorena Barba, Scribe: Greg Newman

Question at hand: Too much: costly; too little: sloppy

B.1.1 Topics in this area that need attention, and why?

• Cost of reproducibility • Level of abstraction • Abstraction level of specifications • Numerical reproducibility versus bitwise • Serial versus parallel regression tests • At what level is appropriate • Reproducibility of science versus exact numeric results • Workflow reproducibility also important • Replication versus reproduction • Value of validating the science with a new experiment instead of exact reproduction • Is there a correlation between exact reproducibility and result quality? • What is the appropriate technology for different domains of science? • Repeated measurements in experimental science • What are the sources of variability? For example, sociology results based on survey data. Ecology

results are also difficult to reproduce precisely • Reproducibility of experiments encourages them to be repeated • Stochastic versus deterministic problems • Role of co-created just right levels • Individual responsibility to make sure own work in reproducible and for how long. Libraries change; may affect reproducibility of my code • Continuous integration notion in open source can be useful • How long is it needed to maintain (longitudinal maintenance of code and third party libraries) • Way-back machine example for web page archival • Is NSF willing to invest in maintaining reproducible software and data storage over a long period of time? • How long? Each domain to determine for itself? Guideline? • COST factors important • Reproducible by whom (usability issues and incentives community specific) • Interpretation norms annotated as well... • Lack of reward systems... but there can be punishment • Cost of support for others down the road... • Competition as barrier • Replication of science • Badges/certifications for doing the replication? • Pubs to publish full replication studies • Metrics of achievement, success, impact • Common sense needed for each domain • Ask question - how close do I expect these results to be reproducible in the future? • The time window of reproducibility for each domain is specific to each domain • Cultural norms of peer review can be created for open sharing of raw datasets etc. • Change in journals and funders community-driven • Some bottom up needed • Cost-benefit analyses

Summary points:

- Domain matters
- Driven by science reproducibility versus immediate results
- Distinction between reproducibility and replicability
- Community driven / domain driven levels needed

B.1.2 Resources and learnings that might be useful to people

- DataONE workflow tools and metadata documentation
- The vast machine (book about climate change science)
- Reinventing discovery (book)
- A provocative article arguing that lack of reproducibility is a positive for science: https://blogs.scientificamerican.com/guest-blog/the-replication-myth-shedding-light-on-one-of-sciencee28099s-dirty-little-secrets/ (Mendel's experiment is thought to be a classic example of cooked data, but it advanced the field of genetics immensely)

B.2 Reproducibility purely based on publications? Can publication mechanisms help with reproducibility?

Moderator/scribe: Kyle Niemeyer

Participants: Yung-Hsiang Lu, Amarda Shehu, Bryan Heidorn, Philip A Wilsey, Mark Ghiorso, Reuben Budiardja, Selcuk Uluagac, Sandra Gesing, Alex Travesset, Sicong Liu, Inanc Senocak, Kyle Niemeyer, Gerry Puckett

Questions:

- How to do reproducible research with (e.g.) export-controlled research (DoD), or sensitive data?
- What modifications can we make to publication workflow to support reproducibility? (page limits, shallow data, software publication)
- Difficult to publish *complete* reproducible analysis, due to external dependencies. Related citation issues, and how to enforce?
- What is the problem? To facilitate research so that other researchers can follow the work? To prevent fraud?
- What is the ultimate motivation behind reproducibility? Identify errors, plagiarism, or more?
- How can mathematical expression be translated to code systematically?
- Is Docker the solution? How about performance-critical research that could be interfered by virtual machines?
- Is it sufficient to verify reproducibility at time of publication?
- Will/should ACM/IEEE issue policy about reproducibility?
- Will the requirements for reproducibility slow down innovation and progress?
- ACM Transactions on Modeling and Computer Simulation (TOMACS) ... reproducibility of simulation results
- Replicable vs. reproducible?
- "Market-based" approach, as we assign implicit scores/prestige to publications and work
- Intellectual property

Possible solution: Analogy to NSF Data Management Plan, where there are no explicit requirements, but "market" forces that pressure PIs to make results more open like competitors.

B.2.1 Topics in this area that need attention

- Benefits of reproducibility need to be made clear.
- Consensus: forcing people to do something is a no-go. However, by people starting to do more reproducible work and share associated products, market forces will tend

to lead to others following similar practices.

B.3 Specify and verify reproducibility attributes of contributed SW components following the open-source model? Who enforces?

Moderator: Namu Patel, scribe: Daniel S. Katz

B.3.1 Topics in this area that need attention

- What are types of reproducibility attributes for general software? (what does this group's name mean?)
- What does reproducible mean? (does it mean the exact same results, results that are within some error bounds or precision, stochastically the same, ...)
- To what extent should software be reproducible? Who is the burden of maintaining on?
- What is the meaning of reproducible software? Experiments that use software can be reproducible, but what does this imply about the software itself? (e.g., can a pipette be reproducible? Use of a pipette? Use of a pipette in an experiment?)
  - Scientific (physics) vs. computer testing (precision, scaling, architectural dependent)
  - Validate on a standard suite of problems
- What benchmarks (linear problems) should be included for reproducibility? Is this sufficient for checking if nonlinear problems may be reproducible?
- What is ideal? (even if not possible today, then how close could we come)
- Understanding inherent complexity; distinguishing from accidental complexity (Fred Brooks)

List of reproducibility attributes:

- Model correctness: Specific the computation in a way that lends itself to knowing that the series of abstract steps for performing the computation are correct (then also need to verify that are the steps coded correctly)
- Sufficient documentation (enough that explains in plain language what the software does and how, provides some simple examples)
- Publishing all data sets, workflows, etc. that go with a publication
- Systematic set of benchmarks that verify correct results (with understanding of what correct means in the context)—key is to know what the tolerance is for correctness
- Platform portability
- Formal method of communication (similar to Aquarium) between model creators (and their expectations) and the team that implements the work (i.e., programmers)
- Software complexity is only inherent complexity; there is no accidental complexity (c.f. Fred Brooks)

B.3.2 Resources and learnings that might be useful to people

- *Software Engineering for Science* (Carver, Thiruvathukal, Hong) https://figshare.com/articles/WSSSPE/830442, https://www.crcpress.com/Software-Engineering-for-Science/Carver-Hong-Thiruvathukal/p/book/9781498743853
- *Fundamentals of Verification and Validation* book by Pat Roach: https://www.amazon.com/Fundamentals-Verification-Validation-Patrick-Roache/dp/0913478121/
- *Verification and Validation in Scientific Computing* book by Bill Oberkampf: https://www.amazon.com/Verification-Validation-Scientific-Computing-Oberkampf/dp/0521113601
- *The Mythical Man Month* by Fred Brooks

B.4 How can we incentivize community participation? (e.g., rules of engagement and incentivization)

Moderator/scribe: Jed Brown

Discussion:

- Chris Rapier: I put software out (HPNSH). Why no response? No developers, etc. But lots of users. Continued maintenance.
- Survey: what do people want to see? But no feedback.
- DK Panda (MVAPICH): Widely used. Focus on quality, production. Get patches from many centers (mostly bug-fix/portability, features from some companies); recognizes contributors.
- How to control quality? Keep extensive test suites, public dashboard, code reviews. Contributors also provide their own tests.
- Moving to Github increased number of contributors by 4x. Eliminated closed group mailing list, all done in users mailing list.
- Frank Löffler: moving all bug reporting and code review to open platform encouraged contributions.
- Nanohub components all indexed on Web of Science.
- How often do we write a paper about software? Very hard to publish software papers (in good journals/conferences). But many software journals coming out. *Geoscientific Model Development*
- How to incentivize upstreaming versus forking? License (without "legal requirement") or otherwise encourage? Upstream good for maintenance, then "we" maintain it and support it. But forks good because upstream may not understand user requirements or have the capability to test/integrate. (Could it be done in a plugin?)
- Sanjay Kale: How is contributor funded? NSF Software Institutes provide some of that role, NSF gives supplements to developers/users (short term).
- Effective ways to build a community? Social media,
- How to monitor that software is being used and not just downloaded? Interactivity

helps,

- Janos Sallai: Starter project templates (easy start, and helps track active users).
- NSF unable to fund maintenance. Charter is to fund innovation.
- Experimentalists incorporate facility costs into proposals. But software users assume software is free.

B.5 How to foster/enforce best practices despite loosely-coupled development approach of scientific software?

Moderator: Paul Bauman, scribe: Abani Patra

B.5.1 Questions that arose:

- Norman: The Question!!
- Norman: Let us share disasters!! Share successes
- Mike: Pose things as question.
- Jim, Drexel: Testing is vital – coverage; Testing harness » Base code; How to enlist community participation?
- Patra/Bauman: How do we engage community? How do we get users to report bugs and features?
- Wang: What is a best practice?
- Tracy: Data Carpentry: How do we define good? When we value research productivity and not software quality?
- Marzouk: What best practices are universal?
- Good enough?
- Define loosely coupled and how is it a problem? How does it differ from tightly coupled?
- Turk: How do I as software developer (lead) – enforce best practices on contributors?
- Mike: What type of software review is useful for tools in well defined organizations as opposed to loose federations?
- Randy: Is there a mapping between best practices and currently used tools?
- Abani: Training and dissemination.
- Abani: Good scientific software differs from general purpose software, in that it highlights the problem you're trying to solve
- Abani: Tradeoffs between transparency and performance

Best practices:

- FAIR principle
- Sohrab: Good code but not accessible! Electronic structure calculations – monolithic and not reusable codes. Enhancements possible ONLY by tightly coupled team – needs personal knowledge.

- MODULARIZATION is a GOOD PRACTICE and promotes reuse by loosely coupled teams. Who is going to pay the graduate students to write tests and do development? AMBER is a good example where there is some modularization and reuse.
- Richard B.: Kernels driven by well defined mathematical constructs and algorithms.
- Mike: Distributed Software Development = "Loosely Coupled"
- Using Distributed Version Control tools is a BEST PRACTICE enabling Loosely coupled development. Social dimension?
- Training tools for promoting best practices are missing. Curriculum? Training materials? Formalizing training in ... Certification processes for developers, students(?) Certification for the software – will promote better codes and coding practices.
- Peer review as part of the workflow. Students and developers adopt it once they see it. Leaders need to tightly maintain quality control and distributed version control allows this. Graduate students can review other graduate student software.
- Software Sustainability Institute guidelines. (https://www.software.ac.uk/resources/guides-everything) Do not separate too sharply between science and developing scientific software.
- CI frameworks –promote consistency and avoid bugs. A best practice. Travis, etc., linked in to Github to facilitate testing. Does anybody resent automated testing?
- Good coverage?? Tools that test the tests.
- Editorial Function: Third Party Developer Documentation is often missing.

B.5.2 Resources and learnings that might be useful to people

- "Good Enough Practices in Scientific Computing" paper https://swcarpentry.github.io/good-enough-practices-in-scientific-computing/
- Best Practices for Scientific Computing http://doi.org/10.1371/journal.pbio.1001745
- Training: *Introduction to Scientific and Technical Computing*, ISBN-13: 978-1498745048 ISBN-10: 1498745040

B.6 Can we circumscribe best practices for new/legacy software for porting to new platforms/languages?

Moderator: Matthew Knepley, scribe: Ganesh Gopalakrishnan

B.6.1 Topics in this area that need attention

Code changes happen in response to platforms and compilers. Risk in moving to newer language/arch must be assessed.

Questions that arose:

- Perf portability verification.

- Engineering study, need a performance model so that we can take decisions wrt whether there are gains in porting
- Vendors avoiding the library porting issues versus vendor lock-in that might rely on broken standards or lose perf down the road
- Tension between specialization and maintainability
- Annotations to best help practices for data race checking such as sharing info and locking protocols. Education issues in this issue.
- What solutions can a small group rely on, given that one cannot "outsource" wrt component support?
- How do groups take decisions in absorbing a new feature such as C++14, versus the downstream risks that they may not be able to assess?

B.6.2 Resources and learnings that might be useful to people

- Performance portability verification before adopting new platform are needed. But this must begin with portable performance evaluation tools. This needs attention.
- How will users be able to reckon with existing documentation and decide how it will port to a new platform (current solution is a person/expertise is associated with each platform)
- On one hand, manufacturer's solutions help avoid library porting issues but has the downside of vendor lock-in that might rely on broken standards or lose perf down the road ingest bugs.
- Sometimes, one gets locked into buggy implementations for which more sturdy open-source equivalents may not be available (e.g., MPI libraries)
- How do groups take decisions in absorbing a new feature such as C++14, versus the downstream risks that they may not be able to assess. Small groups have harder issues in this area.
- Annotations to best help practices for data race checking such as sharing info and locking protocols. Education issues in this issue.
- Are containers a good model?

B.7 How does reproducibility in software artifact creation differ from reproducibility of lab procedures?

Moderator: Eric Klavins, scribe: Frank Timmes

Participants:

- Miron Livney, Univ Wisconsin, pegasus
- Paul Navrátil, TACC, vis scientists
- Katy Borner, Indiana University - data mining vis , data analytics workflows
- Tom Chesthamatham, Utah

- Manish Parasher, cyberinfrastructure oceanography research in HPC space
- Dane Morgan, Univ Wisconsin computational material systems
- Larry Frank, UCSD, Data mining and vis packages
- David Schloan, U of Chicago, archaeologist, integrate data across projects, more sophisticated tools. Heuristic iteration. Capture master-apprentice training.
- Mike Dixon, NCAR; lidar, remote sensing. Image processing on 3d. Current pseudo-manually by scientists. Si2 project is develop toolbox of sequential modules. Scriptable procedure.
- Frank Timmes, ASU; interest in portability to different fields, esp software development. Can master-apprentice system be applied to software development, theoretical work? Astronomy: each observatory had different lens, repeat measurement is costly–same number for $30,000.
- Rion Dooley, TACC; reproducibility on computing calculations

Question: What goes wrong in protocols? Drop test tube, machine breaks, etc. Answer: Cancel operation and try again.

B.7.1 Topics in this area that need attention

- How does reproducibility in SW creation fundamentally differ from reproducibility of lab procedures (where the human element plays a huge role)?
- Where do existing workflows and provenance tracking methods (with sufficient usage instructions and training material) still lack when it comes to capturing lab protocols? GitHub.
- What are some key impediments in catering to the needs of various target groups through a codified "Operating System" for lab protocols? Will it equally work for (say) Astronomy research as for Biology?

B.7.2 Discussion points

- Domains with lab bend - protocols work. Software a work in process - docker containers.
- Does the model work in software development? Teaching coding, cut & paste, co-coding (two people work on one code - twice the cost. another approach is a scribe watching the senior-junior relationship. Sorta like a live coding review. Coding steps unique? Coding is an Art form - Darwinian.
- Miron: reproducibility is mandatory for NIH proposals. It is a meaningful buzzword, like sustainability and cloudy until the rainbows come.
- Is reproducibility a buzzword?
- Is such productivity scripting a Charlie-Chaplin-esque production line with students learning less? Question is where the abstraction level. Disruption. No assembly line

workers.

- Larry: co-coding too expensive. One brain data set all workers use (not phantom). Algorithm development tests. Should start with numerically generate phantom (standard, pristinely clean) dataset - not with brain data! So that benchmarking, apple-to-apple comparisons. Larry is generating several numerical phantoms for testing purposes. Katy would like to use them in teaching IVMOOC.
- Eric: Hardware hard to integrate into workflows - private API. Ideally, we could teach people lab skills like Rosetta.
- Dane Morgan: Code people's behavior—eye opening. Very structured. Informatics skunksworks - undergrad education - run neural network on sine curve, etc., initial training much like biolabs.
- Incomplete data sets; weather lidar, brain data; how to reproduce.
- Katy: algorithm comparison much needed and done in many sciences as part of conference competitions. Model organisms serve as another way to have many use the very same dataset.
- Larry: Action Item: create library of numerical phantoms (well documented datasets) for testing purposes.
- Repository for test suites. Peer reviews for community push on reproducibility.

C Detailed notes from breakout discussions: audience-proposed

C.1 Software metadata: discovery, credit, preservation

Moderator/scribe: Kyle Niemeyer

Participants: Kyle Niemeyer, Cameron Smith, Barbara Lerner, Mercè Crosas, Rajiv Ramnath, Mike Hucka, Ilya Zaslavsky, Abani Patra

Metadata for discovery may differ from (and be more extensive than) the metadata needed for citation.

Need for general tools for recording provenance of computational results, to go along with software citation?

C.1.1 Resources and learnings that might be useful to people:

- Software Citation Principles: https://doi.org/10.7717/peerj-cs.86
- CodeMeta (SI2-funded effort): http://codemeta.github.io/
- Journal of Open Source Software: http://joss.theoj.org/
- Paper with survey on desired metadata: https://arxiv.org/abs/1605.02265
- Recipy (provenance recording in Python): https://github.com/recipy/recipy
- Find out what's important for "making software count", similarly to the project for data "making data count": http://www.nature.com/articles/sdata201539

- Extend current Schema.org for software: https://schema.org/SoftwareApplication

C.2 Sustainability of projects after grants end. The perspective of projects, which mixed federal and private/industrial funding

Moderator: Mike, scribe: Bryan Heidorn, Dan Katz

C.2.1 Discussion topics:

- Astrophysics were supported by grants
- Community software supported on science grants with a few grants for development money. Always worried that the child would die. Open Software helps if there is a developer community.
- When should a software project go quietly into the night? (15 years?)
- Maybe the community wants access to a set of functions rather than an application.
- Maybe the age of software does not matter but evolution may mean a new rewrite.
- Sustaining keeping the original vs sustaining being keeping the functionality.

C.2.2 Topics in this area that need attention:

- Constant updating needed (e.g., security issues, adaptation to evolving software and hardware environment)
- Research faculty and staff end up locked into maintaining software that the community needs. The community finds useful but no funding. Open source may not have enough programmers behind it. SSH - used but why do programmers not step forward.
- Service model - base version plus for fee service for more features.
- Idea: NSF gives software tokens with research awards, awardees give the tokens to software projects where they use the projects; projects can use the tokens they get to argue to NSF that they are needed
- RedHat model: free for academia but charge for service from for profit
- Go Fund Me or Kick Starter? (see guides below for examples)
- Would NSF support budget lines for software support. (NSF would, but another question is what the peer-reviewers would say)
- In scientific code development there needs to be users who need new features or platforms and will do the work.
- May need to training the domain scientists to do a "little" software development - sustainability.
- Programs for technology transfer were briefly discussed, such as NSF STTR: https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504857
- And NSF SBIR (https://www.nsf.gov/eng/iip/sbir/home.jsp)
- And NSF iCorps (https://www.nsf.gov/news/special_reports/i-corps/)

C.2.3 Resources and learnings that might be useful to people:

- Nadia Eghbal's report: Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure (http://www.fordfoundation.org/library/reports-and-studies/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure)
- A handy guide to financial support for open source (
- https://github.com/nayafia/lemonade-stand)
- A guide to sustainability models for research software projects (https://github.com/danielskatz/sustaining-research-projects)

C.3 Near future concern: to make a self-sustaining software package after 5 years. A robust discussion and prior examples showing how to go about it.

Moderator: unknown, scribe: unknown

C.3.1 Questions:

- What examples are there of open-source self-sustaining software after five years?
- What are some successful sustainability models?
- How do you stop software from being out there and stagnating? Keep modernizing and alive?
- Once we put a software out in the open, how does it magically become self-sustaining?
- What is the degree of utility needed from a software "out there" that makes it self sustaining?
- What are marketing/advertisement/getting the word out successful ideas?
- How can a PI transfer ownership to community? Or keep ownership but make effort continue past funding?
- Actual examples from crowd where people are working on sustainability?

C.3.2 Discussion:

- Paul Butler: Involved in two software development projects both in the area of providing tools for researchers to analyze small angle scattering data. Experience with long term sustainability is on the older project that started as a part of the NSF DANSE project in 2006. It managed to expand to an international collaboration with no direct funding for a number of years. Currently it has a growing community of developers, some bringing in short term grants. However I would say the jury is still out regarding its long term success.
- Paul Bauman: Software solution that did not exist was obviously very useful so once written it was a hit. How is it sustained? SI2 project is making this library more useful. So the SI2 project enhances this software (libMesh).

- Bernardi: new code for electron dynamics. Using version control with schedule for releases. Efforts to train users (schools/workshops) are important ways of building community.
- Marzouk: Use software as a teaching tool in other teaching contexts (e.g., summer schools, graduate courses)
- Outreach: go to conferences trying to convince people to submit their software along with their conference abstract. This way this will promote the softwares that are downloadable and stable.
- Westerink: has community, listserv, 20+ questions a day on usage. Everyone working on this gets their own grants, one person doing version control is funded separately.
- Don't open till it is stable not to scare away users!
- Marsden: have worked hard on improving the software, but in the end of year 5 will have to support the software via other types of grants. There are no examples in this community of non-grant supported software development.
- Reach out to colleagues or companies to use the code, but how else to do it?
- Kale: NAMD since 1996: funded by NIH resource. NIH views software as a resource, and funding this software so it is supplied to the scientific community. Charm++ was funded via science funding as part of a team. Now have SBIR to try to sustain it. There are so many people depending on this charm++ software it is scary if it is disappears. The science parts are on grants on and off, but once they are gone no way to fund the CS folks and computer scientists/programmers... hard to create "justification".
- Maechling: software has supported both NSF and industry support. NSF funding gave validation activities and software development. A few years of users checking results and hence very well validated compared to competition. Don't just add features but show it is stable and valid.
- Miller: initial DOE funding (18 years). Now SI2 funding. Now have outside developers (livermore, redhat, etc.) Beyond trying to get the next grant, to go community development. Hard but valuable.
- Newville: X-ray spectroscopy software: SI2 grants + facilities grants. The software is part of the "facilities" so it is funded, but more direct (instead of backdoor) preferred.
- Good way to shift control from senior to junior researchers via joint grants/projects.
- Cheatham: do charge people to use, but never enough $$$. Must use volunteers and grants to fund the software. Can any of us do sustainable software in academia? Google or NVIDIA have money coming from other sources to do "science" on the side.
- Started off with a 3D image processing software, has been successful. Wrapped in other applications. Grew organically. In SI2 pulling back the new methods back into library to augment and enhance software for next iteration of science.
- Condor software 30+ years. Almost all funded by government funding. Maintained by telling funders that if they close us down we will put their phone number on our

web page :-) Key challenge in sustainability: keeping the PI interested (and not the funding). If you stick to something and do it for a long time, you will prevail.

- Rutgers: has built many softwares that were useful software but hard to sustain long term. No good answers.
- Princeton: leverage facilities funding, but still something lacking in this sustainability model in last 20 years. How to make it a dynamic, open system for easy outside input?
- NSF pointed to HDF5 example as a success. Have a foundation. But they got a huge NASA grant. PIs got grants but got a foundation/entity set up to run this thing: get donations, write grants, enhance software, ...
- Educating the next generation is key: not just how to use the code but what happens inside.
- Bauman: Having a low barrier of entry to the software/library is critical. Also, open development model where there is no "dictator" or "moderator". A cohort of experts helps perpetuate the software. Giving warnings on upcoming changes is important too to not tick off the community.
- Who can touch mission critical code is not a light matter. Linux: anyone can write a driver, but who can touch the key kernel parts is a very small number. A great deal of time can be spent fixing problems created by outside contributions if one is not careful and the "dictator" is not playing a strong role. You do need a benevolent dictator but it can be distributed to some extent.
- Miller: some models have a single dictator who has high level vision and keep things going.
- One approach: built a strong and well performing base set of functions (good engine) and ask people to use that and build on top of that (not necessarily to keep changing the engine and trying to improve it). This approach reduces maintenance/fixing costs greatly.
- Success = many people depend on you, all on your shoulder! Commitment is important and vision and willing to commit through. Life commitment to sustain software.
- Did you survive or did you sustain?! :-)
- What are we sustaining? People, software, capability, staff, PI? Need to sustain a PI with vision for sure, and people are key. Different models: support PI, or not and try to get it diffused out? Capability is important for broader community? Say we have five codes in a community: why do we need 5? Competition does help each project become better.
- Education and training: we need to teach the incoming/young students to have better software writing skills. This will help sustainability (no matter how defined).
- Dan Katz has a git account with ideas and examples of sustainability. "Who are you going to charge?" is the final distillation.
- Some of the proposed solutions are for large scale projects; for medium to small sized projects not clear they can benefit from those approaches.

- How should we/NSF decide what *not* to sustain? What is the process for selecting for deciding not to sustain something? How many versions of the same thing do we need? How much duplication of effort do we need given limited resources? We don't have a way to deal with this hard question.
- Why do people reinvent and remake software? (a) Doesn't do what they want. (b) Doesn't run due to it not being really maintained. (c) The available software capabilities are not distributed (are being "held back" by owners) so you just have to do it again yourself.

C.4 Extensibility in Numerical Software

Moderator: Namu Patel, scribe: Ed Valeyev

C.4.1 Questions

What does "extensible" mean?

- Augment/extend part or whole of an algorithm + data structure.
- Allow user to inject functionality into existing core.
- Have to distinguish extensibility and reuse ...
- Portability as a kind of extensibility
- Ease of adding features, i.e., extensibility = 1 / Incremental cost to add a feature
- Maintainability
- Extensibility is related to ability to add major features, i.e., extensible software must allow features that were not supported by the original design.

Languages in the room:

- C++
- Python
- C
- CUDA

C.4.2 Topics that need attention

Domain Specific Languages (DSLs) as a way to extensibility

- Example uses in tensor computation
- Some examples of building DSL using LLVM infrastructure (OpenMM); LLVM can be an onerous dependence
- FEniCS is another example of DSL use; it's a finite element code with low-level kernels get generated at compile/runtime (if in C++/Python)

How to achieve extensibility/common traits to extensible software:

- Common traits: data abstraction, modularity, factorizable design (separation of data representation, algorithms, etc.)

- Extensibility can be improved by optimal design of class interfaces (e.g., overly-intrusive API, e.g., classic setter/getter, can prevent changing implementation details later)

C.4.3 Resources and learnings that might be useful to people

Success-story/best-practices examples:

- PETSc
- IBAMR
- Eigen
- Moose (more successful at extensibility, although at some cost of performance)

Useful resources:

- Jed Brown, Matthew G Knepley, and Barry F Smith. Run-time extensibility and librarization of simulation software. Computing in Science & Engineering, 17(1):38–45, 2015.

C.5 Education for software development and release

Moderator: Tracy Teal, scribe: Yung-Hsiang Lu

Participants: Ganesh Gopalakrishnan, Andy Miner, Gogan Agrawl, Amarda Shehu, Anton Van der Ven, David Tarboton, Randy Heiland, Tracy Teal, Anthony Aufdenkampe, Upulee Kanewala, Maggie Myers, Matthew Turk, Mike Sokoloff, Sicong Liu, George K. Thiruvathukal, Yung-Hsiang Lu, Emanuel Gull, Lorena Barba, Katy Borner; Daniel Katz (not in room, but interested in follow-on)

C.5.1 Problems

- Difficult to create a new course (especially required) about software engineering
- Who should "certify" the essential skills needed for software?
- Professors are expected to cover a lot of concepts in textbooks but students may not understand the materials
- Students not in computer science/engineering cannot enter software engineering courses because there is no space

C.5.2 Possible solutions

- Short courses (a few days, possibly in summer) for students that are not in computer science
- Certification of training
- Instead of teaching many possible solutions, pick one or two so that students can master one or two before learning other possibility

- Use education modules made by others
- It is possible to transform "traditional" chalkboard course to become computational. Students have to submit their solutions through Github. Students can learn abstract concept, used to be purely mathematical, better through the interaction with software. Computing and software as a form of representation. Calculus is a representation of concept, so can software be.
- Start with small groups of undergraduate students. Train them to build the materials. Use MOOC

C.5.3 Topics that need attention

- Reward system (incentives) in academia for releasing useful software
- Software carpentry or other short-term (a few days) intensive training not the same as semester-long courses
- Balance between knowledge for software engineering and the time to learn domain knowledge

C.5.4 Training in the gaps: ways of doing this

- Short courses
- Two or three day bootcamp style workshops
- MOOCs
- Integrating teaching these skills into domain specific courses
- Research practices type course

C.5.5 Resources and learnings that might be useful to people:

Success-story/best-practices examples:

- Uncle Bob videos: https://cleancoders.com/—good intro to software design, developed originally for apprentices in software shops.
- https://software-carpentry.org/, teaching researchers in science, engineering, medicine, and related disciplines the computing skills they need
- https://www.edx.org/course/linear-algebra-foundations-frontiers-utaustinx-ut-5-05x, Linear Algebra - Foundations to Frontiers
- MOOC, Coursera
- "Practical Numerical Methods with Python" (aka Numerical-MOOC) MOOC on Open edX and GitHub — five modules based on Jupyter
- AeroPython on GitHub: https://github.com/barbagroup/AeroPython
- CFD Python on GitHub: https://github.com/barbagroup/CFDPython
- Barba-group "Essential skills for reproducible research computing"
- https://www.codecademy.com/
- https://www.codecademy.com/learn/javascript

- https://www.coursera.org/browse/computer-science?languages=en
- NSF grants to create short courses for domain experts
- Resources for every SI2 grantee can benefit

C.6 Cloud services for software and data

Moderator: Unknown, scribe: Janos Sallai

Cloud services: Cloud-based service are mainstream in commercial software. Software containers and virtualization help reproducibility. Today companies are much more open to computing in the cloud than a few years ago.

Managing storage:

- Managing storage is of key importance. What happens to the data if the project funding runs out? If storage bills are not paid, data will be deleted. What if others need the data later?
- Cloud storage providers (e.g., Amazon) have very high egress charges.
- CITRINE – a private company doing data storage for the materials and chemical domains. Customers need to pay if they want to keep their data private.
- Unsolved problem: fully available storage for everyone. Science gateways today do not have data solutions.
- Challenge: making data discoverable and accessible.
- It is important that proposers budget for storage costs in the proposal budget, because it is part of the cost of doing research. The problem is that it is hard to budget for that if the future usage and storage requirements are unknown in advance.
- NSF could help with providing a storage facility for long term data storage of data. Or should data be managed by the universities? Libraries?

Lack of awareness of public resources: Public clouds. Some of them are fully utilized, some are underutilized. Restrictions on usage (e.g., 24-hour limits). Public data storage resources.

C.7 Growing the user community, best practices

Moderator: Unknown, scribe: Katherine Lawrence

Participants: Luke Wilson (TACC, volunteer computing w/in national research infrastructure), Dane Morgan (U of W Madison, molecular simulation/materials), Michael Bell (Colorado State, Remote Sensing), Tim Menzies (NC State), Peter Cummings (molecular modeling), Jerry (Quantum properties of materials), Shaowen Wang (UIUC, Geospatial [GIS]), Jakub Kurzak (U Tennessee Knoxville) Jing (UNC Charlotte, urban trajectory

visualization software), David Irwin, Sameer Shende, (U of Oregon, User software expertise), Eric Klavins (Trying to manage the growth of the community and not pissing off as we grow), Perry de Valpine (UC Berkeley, computational statistics [applied scientists/end users, research statisticians]), Tom Quinn (UW, Computational astrophysics), Derek Richardson (U Maryland, astrophysics), Frank Timmes (ASU, astrophysics), Paul Navratil (TACC, visualization), Gene Cooperman (Northeastern U., transparent checkpointing in user-space)

C.7.1 Topics that need attention

- Eric Klavins: Does anyone have someone who specifically manages their community?
- Frank: May depend on the size of the community–what size is yours?
- Eric: We have about 100 of self sufficient. Fewer high-maintenance, lots of work but valuable.
- Jakub: Hard to judge the size of ours.
- Perry: Interested in non-virtual outreach to users. Hard to know what works well and not. We wrote grant to invite key users to our university as key disseminators.
- Frank: Two mechanisms: Annual summer school (not subsidized, has a waiting list to get in; bootcamp; people act as ambassadors to their campus); Ambassadors (person goes to the campus to interact for several days)
- Sameer: Offer workshops; also tutorials at conferences; Now other institutions are teaching our tools.
- Michael: We're building a new tool for a field that has existing legacy tools. How do you deal with the inertia of getting them to adopt. The old tool users are mostly retired.
- Perry: We had a similar situation, and the old tool creators were willing to list us on their website.
- Sameer: Is your format open? Can you maintain compatibility with earlier tools so that they can submit their data without reformatting? (Or offer a conversion tool?)
- Jakub: How do you go about marketing a new product? Great if you can convince others to advertise (e.g., Intel newsletter)
- Tim: What are the user stories that you can share? Can people post enhancements that are sorted
- Jakub: How do you boot up from ground zero?
- Eric: I only wanted users who could support my research. I wanted to meet with the person f2f to develop those relationships that matter. Hope that the effect beyond that is that it will spread by word of mouth.
- Tim: Most successful open source software does something you care about (Raymond).
- Shaowen: In the area of GIS, we were advised don't worry about the existing tools (commercial) and let people discover the value.

- Shaowen: Question: As academic researchers, we get to a point in our scientific inquiry. We have 1000s of people approaching us, but we're not designed to work with that. What are the strategies for going from here?
- Tim: Did your developers start by establishing connections to the user community? E.g., Github listing to show that you are responding to their concerns.
- Shaowen: Our users have been part of our requirements analysis, but our users are not computer geeks. They are not interested at the coding level but the GUI.
- Perry: Some users are not going to ask that level of question. Only using email or the user mailing list.
- Tim: What are the examples that the users are requesting a feature and in some amount of time, there's a response to that request.
- Frank: What's your goal?
- Shaowen: NSF is interested in the community angle and the broader impact. There are a lot of software features that need to be created. Spectrum of evolving what you have. Could convince industry or your university to set up a system to deal with this.
- Eric: Has anyone started a company to do this?
- Sameer: Academic project that expanded to the point that we needed professional support.
- Eric: We're not sure if the market is big enough to warrant a company.
- Frank: In your case, there seems to be a market there.
- Perry: Do you make money on the workshops you offer? A: Cost neutral.
- Luke: Experience with converting the user community into a developer community?
- Shaowen: Very difficult.
- Eric: I'd love to know that answer. We get big requests.
- Tim: Turned his community into programmers by using their words and created 50 rules, and then they added another 400 rules.
- Eric: So to generalize, if you give them the language and some stories, they can run with it.
- Perry: I've had some situations where we've had a beginning user that is using the software to lead to publication. I've been able to connect them to a more advanced user who has the incentive of contributing to research. This helps them bring up the newer user into a developer.
- Eric: Documentation. Ours is okay. I find the new students don't want documentation, but if I give them a video, they'll watch it 3x. I record what I'm doing on the screen to show what I'm doing.
- Tim: Documentation doesn't sell software; it is the convincing story.
- Perry: I find myself putting more effort on the core features, not the documentation and dissemination.
- Frank: I get the impression that your projects are one PI with an assortment of students and postdocs.

- Tim: Putting stuff online is fuel that helps me stay ahead of the pack. We've trivialized work for centuries, not rewarding the work that goes into the software that's at the margins of the slick products. It isn't a little detail–it isn't MS Word; we're building it.
- Sameer: What do you do about having research programming staff with a career trajectory?
- Eric: The majority of people at this event are not computer scientists, so their students are trying to get a degree in a discipline, not publishing in CS journals.
- Paul: Computer scientists are not the same as software engineers, so adding a CS person does not give you software engineering expertise. CS is more like a discipline. We need to be precise in our language so that people come to the Software Engineering field to hire. Publishing opportunities are available for domain scientists at these computing conferences.
- Perry: There are new journals in domain science fields that are focused on methods or software. The publications are enhancements to a career, but not necessarily tenure getting.
- Michael: Gets back to your point about users who might want to contribute a feature but don't have the background to do it–yet you don't want to discourage them.
- Frank: If you put in hooks, then people can attach their modules to the hooks without messing up the main code.
- Tim: In our dept., needs to be practical.
- Shaowen: Are you saying. . . ?
- Tim: Do you know green engineering? Optimizing Shaowen:
- Tim: For many years, astronomers had to grind the lens, and eventually they could look through the lens to do the science. We're in the same place with software.
- Peter: Spoiled by the Institute for Scientific Software Engineering, 100 employees and 6–7 faculty. We partnered with them to recast what we do in the way that they do things.
- Dane: Does seem like there's an opportunity for matchmaking between the domain communities that want help and the software engineering communities that want projects.
- Tim: Watch out for the programming language mafia! (Internal to the NSF.)
- Perry: Is there a place for a grant that would connect computer scientists/engineers and domain scientists.
- Katherine: NSF's ITR program in 2004+ was exactly that. The challenge was that the computer scientists had the goal of doing their research, which necessarily delayed the research of the domain scientists. Also, they had to watch their language and terminology to understand each other.
- Tim: There's a language to bridge. I look forward to doing that over the next two days.

D Detailed notes from breakout discussions: Software Carpentry

D.1 Developing contributor guidelines (approaches to improve research and education for better software sustainability)

Moderator: Matthew Turk, Scribe: Kyle Niemeyer

What do folks want to get out of contributor guidelines?

- Growing? Maybe the small model won't work anymore?
- When does the project development cycle break down? The barriers to entry, for instance—when are they too high?
- What is the point where you need formality?
- Level of formality?
- "Who are contributors?"
- Are we talking about programmers?
- Also consider those that create documentation, examples, project website
- How to codify process of people getting involved, at any level

Examples of contributor guidelines:

- https://github.com/nteract/nteract/blob/master/CONTRIBUTING.md
- https://jupyter.readthedocs.io/en/latest/contributor/content-contributor.html

Developer guide vs. user guide: former gives you mental model

- http://palm.muk.uni-hannover.de/
- http://enzo.readthedocs.io/en/latest/developer_guide/index.html
- http://yt-project.org/docs/3.3.3/developing/index.html (http://ytep.readthedocs.io/en/latest/YTEPs/YTEP-1776.html for info on editorial process)
- https://github.com/pr-omethe-us/PyKED/blob/master/CONTRIBUTING.md

Guides

- Style guide: Automate it, w/ clang: http://clang.llvm.org/docs/ClangFormat.html, https://www.pylint.org/, can be included
- How to write a contributor guideline document (could this group write/start?)

General comments/questions:

- How to insert "chutes" and remove "ladders"
- "Opening up can of worms"—how do we decide why things change?
- "You can do validation against reality, and things might get worse or better, but are you doing what you think you're doing?" V & V
- "When do you accept a pull request?"
- "Grand challenge"
- Things that are wanted: inputs, expected results, change.

Call to action:

- Guidelines and starter kit?
- Objective vs subjective
- What to cover
- How to distribute
- How do socialize developers that don't necessarily want contributors, but need them?
- Developing guidelines for "reference" and "how things change," and how to interpret that
- SI2-specific
- Things to ask of the project leader—it's absolutely not a one size fits all.
- "What is a contributor?"
- "How would I contribute in these different categories?"
- Tips/suggestions for things to help encourage contributions (e.g., "beginner-friendly" issue tags)
- Guide/document needs its own contributor guidelines...

Topics that need attention: Need how-to guide/document on writing contributor guidelines.

Partipants/potential contributors: Matt Turk <matthewturk@gmail.com>, Kyle Niemeyer <kyle.niemeyer@oregonstate.edu>, Mike Norman <mlnorman@sdsc.edu>, Inanc Senocak <senocak@boisestate.edu>, Bruce Berriman <gbb@ipac.caltech.edu>, Tom Quinn <trq@astro.washington.edu>, Derek Richardson <dcr@astro.umd.edu>, Paul Butler <pbutler@utk.ed>, Anthony Aufdenkampe <aaufdenkampe@limno.com>, Cameron Smith <smithc11@rpi.edu>, Michael Bell <mmbell@colostate.edu>, Philip Maechling <maechlin@usc.edu>

D.2 Integrating training for developers and users into software development projects

Moderator: Daniel Katz, Scribe: Doug

Participants: Carlos Maltzahn, Douglas Thain, James Shackleford, Bart Miller, Reuben Budiardja, Perry de Yalpine, Matt Newville, Mercè Crosas, Rion Dooley, Peter Elmer

(Training is key component that drives use and adoption of software, but it often isn't included as a part of the project proposal or development. How can we better integrate training for developers (like software development best practices) and users (including effective documentation or training materials for how to use the software) into the software development process and make it a priority rather than an afterthought? Additionally, use of the software may require other knowledge, such as using the command line or initial programming in R or Python. How can we coordinate efforts on training for those foundational skills and effectively provide opportunities or resources to users?)

Transcript: Q: How to train our own developers and contributors to write solid code, documentation, and training materials? What is their motivation to contribute? Obs: A code contribution consists of not just code, but also test cases and documentation and an example. Without these things, the pull request will die.

Q: How do new developers bootstrap, so that they know how and where to contribute? A: One approach is to have a bi-weekly contributors call plus a yearly workshop with training A: We created a template for plugins that others can follow. Obs: Consider the other side: teach students how to interact with outside projects, not just our own. E.g. OS class should how to deal with Linux source code. Topic: Merging of user and developer mailing list.

Q: Do you keep user and developer documentation separate? A: For some tools (frameworks), the users are developers, and documentation needs to be written differently than for other tools. Documentation needs to be written with the knowledge of how it will be read by different groups.

Q: How would you go about reflecting these ideas in a proposal? A: Dividing effort in students is hard. Large projects can divide work btwn people with different skills. A: Community incentives matter – If we want students to be skilled in explaining things, doc/outreach/training should be part of it. Project metrics matter. A: It comes back to sustainability. If there is no life after 3–5 years, why document? On the other hand, doc/training can extend the functional life of a repository. A: Training is like data management: it's not the objective, but doing it enables you to have greater impact. A: Communication may have greater value than documentation. Obs: Reproducibility may be connected to accountability. (some recent paper says: reproducibility = accountability + transparency) Obs: Documentation can be a recording of what experiment was just completed, to get un-confused.

Q: What makes more sense? Two students code, one student documents; or everyone does both? (Sense is the latter) Obs: We require students who work with a piece of code but have difficulty understanding it to add their own comments as they go. "Breadcrumbs" Obs: use blog (e.g., on Medium) for the developer to describe what he/she is doing, gives visibility into what is changing and why

Q: How to decide between spending time building the new feature versus documenting the old? (Default: build the new!) Obs: Package managers cause people to be locked into particular packages, at least until they need something that's not in the built package. Should developer be trained to get changes into the package build system? Obs: Questions and feature requests often come through the user list - rare that changes start in the code base directly Obs: Use clean, external VMs as the place to test and install your software. Eliminate deps on local infrastructure.

Q: Do you find Slack to be useful in interacting with your community? A: Yes! We have better questions, better conversations, and old users help to train new users. Limitation: Need to mine good users questions into an FAQ b/c (free) slack history is limited. A: Similar experience, we were able to pay for account for core developers, but then have free

channel for large public number of people.

Topics that need attention:

- What are the additional things besides code that need to be part of a contribution? (e.g., test, documentation, an example) How do projects train their contributors to do this?
- How do users get trained on how to contribute? Can a "template" be used? Or a plugin system?
- What about projects that don't come from you, that are already existing and well-established? How to teach students to use those resources?
- How can multiple projects coordinate similar needs?
- Importance of providing the big picture, high-level architecture to train contributors
- Use the idea of developers being able to explain things to guide how they feel about the project. The concept of having to explain things means that you have to think about how the other person will understand it, which means that you do things for them, not just because you have to

Resources that might be useful:

- SI2 project that motivated grad student to explain/teach his software: Big Weather Web (bigweatherweb.org) and Popper (falsifiable.us).
- An example of a useful process on how to write documentation for R packages (same approach can be applied to other type of software projects): https://medium.com/zelig-dev/documenting-zelig-31e7d2f6c11b#.dlrruw1rwhttps://medium.com/zelig-dev/testing-zelig-b41961cbfab8#.5ntfw6cq2
- Slack and Slackin: https://www.npmjs.com/package/slackin

D.3 What are the top-n topics that software developers should know and how to teach them? And when do you need to bring in professionals?

Moderator: Chris Rapier, Scribe: Katy Borner

Participants: Chris Rapier (Pitt SC, degree in history), Richard Brower (BU), Robert-van de Geyn (UT Austin: How to proof correctness of a program), Maggie Meyers (UTexas A, Math, stats, educ), Daniel (Move CS department in School of Computing and Info), Namu (Graduate student at NU, applied math), Michael Dixen (NCAR), Philip (curious), Utah State, Ashish (SRI - provenance for reproducibility), Jing (UNCC, CS, mech eng), Marco Bernardi (CalTech, applied physics – works with many physicists that need to dev professional code). Upulee (Montana State, CS will move into School of Computing, will teach CS for non-CS majors), Katy Borner (Indiana University, engineer, cs, infosci, info vis. How to develop production strength code in academic environment?)

Topics that need attention; tagged as "timely" (TS) and "timeless" (TY):

- Leadership/Management Skills (TY): Shared language is important – IT (security issues) vs. research programmers vs. researchers
- Theory (TY): Numerical accuracy, Algorithmic complexity, Formal correctness
- IT (TS): Security, Resource management
- Data (TS): Data management, Databases
- Software (TS): Software engineering workflow, Data structures, OO, Modularity, HCI, Mining, Visualization
- Quality control (TS): Testing, Debugging
- Scaling up (TS): Parallel computing, Programming for performance
- Productivity/Efficiency tools (TS): Version control, JIRA

While this sounds like two full MS degrees, discussion was mostly about "continuing education" as much of this knowledge/skills changes rapidly yet needs to be kept up-to-date in a timely fashion.

Also consider integrating into workplace. Every course should include some form of computation.

What can be taught in classroom, what needs to be done regularly, as part of lifelong training?

Teach "timely" on rapidly evolving tech/practices via MOOCs. Teach "timeless" knowledge on algorithm complexity, human perception/cognition via classroom courses/degrees.

If possible, require/strongly encourage/support that programmers take 40h or more of professional development each year. This will SAVE money in the long run and result in higher quality code.

Watch out for nano-credentials, micro-masters.

Resources that might be useful: Use open, free MOOCs and other training:

- https://www.mooc-list.com
- https://www.class-central.com
- https://software-carpentry.org/lessons
- http://www.datacarpentry.org/lessons

Consider teaching a MOOC or adding resources: The template for our workshop websites can be found in this repository, and the styles repository contains the CSS, Javascript, and other tools and resources these repositories share.

D.4 If a project involves SW: what training and practices should be required to be described in an NSF proposal?

Moderator: David Tarboton, Frank Löffler

Proposal for answer to Dear Colleague Letter from the SI2 community:

Request for Information on Future Needs for Advanced Cyberinfrastructure to Support Science and Engineering Research (NSF CI 2030), https://www.nsf.gov/pubs/2017/

nsf17031/nsf17031.jsp

Work on this project has been moved to separate Google doc: https://docs.google.com/document/d/1-O2S7jQ1oSGqg9PPsFJIfASrkNKWy2ZB-EylRzK5oDw/edit?usp=sharing

Software is pervasive in research. Software is a research modality, like an instrument. It needs to be done right. Research software use ranges from use of commercial packages or packages associated with instruments, to spreadsheets, scripts and programs that are used in data analysis and modeling that may extend to long running high performance computing runs. Proper use of software is needed for research integrity, reproducibility and trust in the findings. This is equivalent to proper use of an instrument in the research.

We would like to suggest to NSF that wording be added to GPG requiring proposals to address the use of software in a "Software (training) plan" (1 page addendum possibly, maybe merged with data management plan). Such a statement should address

- How will project participants be trained in the proper use of software
- What software will be used
- How software results will be tested and validated
- How does software support research transparency and reproducibility?
- Already part of data management plan:
- What software management practices such as version control will be used
- How will software artifacts be preserved as part of the data for reproducibility

Comparison to existing, similar proposal artifacts:

Data management plan

- Similar: also mandatory
- Dissimilar: does not include training, reproducibility

Post-doc mentoring plan

- Similar: mentions training
- Dissimilar: optional, only about specific group of people

How projects will address topics that some of the other groups came up with need to be addressed. E.g., n CS topics from group 4.

Interested authors (need more): Frank Löffler <knarf@cct.lsu.edu>, David Tarboton <dtarb@usu.edu>, Dan Katz <dskatz@illinois.edu>, Lucas Wilson <lwilson@tacc.utexas.edu>, Nancy Wilkins-Diehr <wilkinsn@sdsc.edu>

D.5 How to design on demand training for a user community

Moderator: Wolfgang Bangerth (Colorado State, Math); scribe: Katherine Lawrence (U. of Michigan School of Information)

Participants: Ganesh Gopalkrishnan, Mark Ghiorso (U of Washington, Earth Sciences). Gerry Puckett (UC Davis, Department of Mathematics & Computational Infrastructure for Geodynamics), Gene Cooperman (Northeastern), Tim Menzies (NC State, Computer Engineering), Dani Brake (Notre Dame, Computational Mathematics)

Topics that need attention:

- Wolfgang: Run "user and developer workshops" every two years or so, pay every graduate student's ticket to visit us. In the last few years, we've set aside 1/3 of time for talks, 1/6 of time for future topics we want to address, rest of the time is open time to work. At the beginning of the workshop, do lightning talks to introduce what they're working on (3 minutes and 3 slides). They end up congregating to work together, and there's a lot of energy. These workshops last 5 days.

  Also now run "hackathons" for people who have been developing our sw for years. No talks. From morning to night we develop software, review each other's code, if you have a problem, there's someone who can answer questions. Has helped us build community well.

  Helps us see that the people we are submitting our code to are real people and mean well. Builds community. They do about 1/3 to 1/4 of annual development during the hackathons. The patch review is exhausting and the tester gets bogged down by the volume. It is a problem that it delays, but could be solved by money. Takes 1/2 hour on average for a patch to be tested.

  The funding comes from our SI2 project for the workshops. The hackathons are funded by a Center in California that is funded by the NSF. Neither of these activities are particularly expensive.

  Workshop costs $50K for transportation, accommodation (double rooms). Hackathons costs $1000–1500 per person for 10 days for approx. 15 people and they rent a house in the boonies, where they can have food and internet. Ten days is a good amount of time because they can really get some work done.

  Also holds workshops around the world that ends up recruiting users and developers.
- Mark: Will be having a workshop of users (not developers) whose research is completely reliant on these complex codes that have been developed by the people who are running the workshops. How do we convert the community from interested users into interested developers? One thought is to wrap the python in Jupyter notebooks. We'd have them set up real research projects and motivate them to develop the code. This is in the earth geosciences. There's a huge number of people who rely on spreadsheets for their research.
- Ganesh: There;s a person doing research on body segmentation (?) doing something along these lines who you should meet.
- Tim: Add the Zepplin into the project, which will help motivate them. Need someone to prototype the snippets. The workshop approach is what you need to get core developers.

  I spend zero money on travel, but we get people who download our code and find our errors. Be the change you want to create in the world. We just do it. There are so many advantages in creating the community that if you just start it, keep kissing frogs to find those who will adopt. Make small releases.

  Other people have the idea of "apps" which are larger units. Smaller increments to

existing stuff will get you more involvement.

- Mark: I have the idea of getting people away from the apps and moving to smaller libraries (in Jupyter notebooks).
- Tim: Or rest interfaces.
- Wolfgang: Do at most 1 or 2 hours of presentations in the mornings. You need a sizeable number of people who know the code to mentor the others (1 out of 4 or 5). Put people together in small groups and see what they can come up with. The experienced people can float around the room. If you have 20 ppl in the room, you need more than yourself to advise.
- Gerry: What works at hackathons is to have a github page with issues and tag them with various categories before the workshop to help people identify what they want to work on.
- Wolfgang: On a sizeable project, you'll have about 100 or 200 issues. We send out the list of the starter projects ahead of time.
- Tim: I've run github for conferences or book development as a way of sharing ideas, not code. The properly conceived "tests" is "science."
- Wolfgang: People show up at the workshops or hackathons because they want to do their science. At the end of the day, we make sure they can consider how they will generalize the code they've created for themselves so that people can use it for other purposes.
- Tim: If you have people who want to add value but are not coders, you can give them tasks such as proofreading documentation or creating a better visual display. Minimize the activation energy.
- Gerry: Anything bigger than a bug fix needs manual entries (all written in LaTeX). Everyone has to submit pull request (to pull the change into the master repository). Can use Git to write papers and it is so cleaner.
- Tim: It is impossible to get big software done, but if you do things in little chunks that eventually amount to something substantial. (See, for example, David Allen's Getting Things Done.)
- Gerry: As a professor, I can ask students to get discrete tasks done.
- Wolfgang: CIG has been running webinars that are uploaded to YouTube. The purely science ones are not so often watched. But lectures that demonstrate how to do something and are interactive, these are watched the most (100s of times). The more theoretical, not hands-on are not so popular. There is great value in watching someone actually do something and modify parameters and seeing what happens. That's something we almost never show in our classes.
- Tim: If you want to build communities, let people do small things. If you've got continuous change to your system and the list of things to be changed are getting marked as complete, it gets people engaged. If someone suggests something that should be done, show them our priority list and ask if their idea should be moved up the list.

- Ganesh: Can I teach a large class the principles of git and then run the class with them using it?
- Gerry: Yes! Git has another feature that lets you send an email notice to something if you select their @name. There's a steep learning curve to some aspects of Git, but you can give them a simple introduction.
- Tim: The best initial approach is to have small pieces that everyone commits to Master.
- Gerry: There's an upstream, master, and your own repo.
- Tim: Can use Git to teach Git by giving them a small paper/website to author together. They'll overwrite each other's work, but nothing is broken.
- Wolfgang: In my project classes, I typically use SVN. You only need to know how to commit, so it is conceptually easier. No branches in these projects. This helps ppl understand version control.
- Gerry: Why even bother with SVN when undergrad sophomores just get Git right away.
- Wolfgang: it is really difficult to document workflows and how it is done. You need to show it (e.g., here is the slider I move, this is the button I click). But to share "What is the function that this does?" should be written down. You can get the big picture by watching through the code and how you work with this.

D.6 How to guarantee a career path for people who are essential but don't want to be academic PIs?

Moderator: Emanuel Gull; scribe: Greg Newman

Topics that need attention: Career paths for staff scientists / students / essential for software project

Questions that arose:

- How to maintain developer roles versus PI - how to compete w/ google/facebook
- How do you keep soft funded model going sustainably?
- How do we provide software developer career path in grant funded environment?
- Independent research units
- How do you sustain the university model itself?
- Where is support for F&A going back to researchers?
- Convince university that research computing is a core facility?
- Salaries are too low! How to remedy this at university?
- Base funding from VPR as an approach?
- How to fight BOTH insecurity and low funding?
- Treat computational people as like librarians
- Morph schools into schools of information and make them core funded
- Research computing is like a library - essential to all disciplines

- Separate research computing from basic IT
- Serious programming is part of the infrastructure
- Ask university to set aside a fraction of IDC serves some purpose but is not the entire answer - F&A / tuition / others?
- NSF to guide universities to better support
- Job security via degree activities
- Build degree programs in scientific computing
- People career tracks... different titles...
- Subsidize 'centers' funded by industry with return on investment of students going to work with them?
- Leverage incubator model and 'alums' approach for philanthropic donations?
- Major Research Equipment (MRI) - need a separate call for computation
- CCStar model - funds innovative tech and campus infrastructure
- NIH core center approach...
- Increase integration of SSI and S2I2 efforts
- **Current configuration is not sustainable for career path development**

Resources that might be useful: TACC example - quasi academia and full industry national lab scenario