2016 Software Infrastructure for Sustained Innovation (SI^2) PI Workshop

Workshop Report

A National Science Foundation Supported Workshop
April 15, 2016

Report from the NSF-funded workshop held February 16-17, 2016, in Arlington, VA for Software Infrastructure for Sustained Innovation (SI^2) Principal Investigators (PI).

This material is based on work supported by the National Science Foundation under award number 1606994. 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.

PREFACE

This report summarizes the organization and execution of the National Science Foundation (NSF) sponsored workshop, “Software Infrastructure for Sustained Innovation (SI^2) Principal Investigator (PI) Workshop,” and the observations and discussions it generated. The workshop was held February 16-17, 2016 at The Westin Arlington Gateway in Arlington, VA and was professionally facilitated by Knowinnovation. It was attended by 98 participants from SI^2 program supported projects, industry, and government laboratories. The workshop served as an open forum for identifying challenges and consolidating common approaches used by software practitioners to organize, manage, and sustain academic research software infrastructures. By facilitating the open exchange of experiences and knowledge, the workshop yielded valuable insights to help inform existing and new software infrastructure projects, and also helped nurture a community essential to addressing upcoming challenges in software infrastructures.

WORKSHOP ORGANIZING COMMITTEE

Frank Timmes (PI)
Professor in the School of Earth and Space Exploration
Arizona State University

Matthew Turk (Co-PI)
Research Scientist, National Center for Supercomputing Applications (NCSA)
Research Professor in Astronomy
University of Illinois at Urbana-Champaign

Stan Ahalt (Co-PI)
Professor of Computer Science
Director of RENCI
Director of the Biomedical Informatics Service
University of North Carolina at Chapel Hill

Shaowen Wang (Senior Collaborator)
Professor of Geography
Associate Director for CyberGIS
Founder and Director CIGI Laboratory
University of Illinois at Urbana-Champaign

Ray Idaszak (Senior Collaborator)
Director of Collaborative Environments
RENCI

Richard Brower (Senior Collaborator)
Professor of Physics and Engineering
Boston University

Chris Lenhardt (Senior Collaborator)
Scientist in Environmental Data Science and Systems
RENCI

Karl Gustafson (Senior Coordinator)
Project Manager
RENCI

SECTION 1.0 - EXECUTIVE SUMMARY

Software is an integral enabler of computation, experiment, and theory and a primary modality for realizing NSF's Cyberinfrastructure Framework for 21st Century Science and Engineering (CIF21) vision. Within this vision, the SI^2 program has the overarching goal of transforming innovations in research and education into sustained software resources that are an integral part of the cyberinfrastructure. SI^2 is a cross-foundation program that generates and nurtures the interdisciplinary processes required to support the entire software lifecycle and promote the successful integration of software development and support with innovation and research. SI^2 currently supports vibrant partnerships among academia, government laboratories and industry for the development and stewardship of a sustainable software infrastructure that can enhance productivity and accelerate innovation.

To sustain and enhance the growing community of SI^2 projects and researchers, the fourth annual "SI^2 PI Workshop" was held February 16-17, 2016 at The Westin Arlington Gateway in Arlington, VA. Building upon the 2015 Workshop,[1] the 2016 Workshop had five strategic objectives that align with the broader SI^2 program goals:

1. Serve as a focused forum for PIs to share technical information with each other and with NSF Program Officers.
2. Explore innovative topics emerging within software communities.
3. Discuss emerging best practices across the supported software projects.
4. Stimulate thinking on new ways of achieving software sustainability.
5. Gather the shared experiences in an online web portal.

These five strategic objectives were embedded in four themes of the 2016 workshop:

1. Training computational scientists.
2. Software organizational forms.
3. Leveraging industrial software development.
4. Domain-specific software infrastructure.

The 2016 workshop was professionally facilitated by Knowinnovation and blended a traditional top-down driven agenda with four keynote speakers (one for each of the four themes) with a contemporary, participant-driven agenda. The workshop was attended by 98 participants from government laboratories, industry, and SI^2 program supported projects, which included recipients of the following grant types: Scientific Software Elements (SSE), Scientific Software Innovation (SSI), Scientific Software Innovation Institutes (S2I2), Rapid Response to Research (RAPID), Early-concept Grants for Exploratory Research (EAGER), Designing Materials to Revolutionize and Engineer our Future (DMREF), Division of Chemistry (CHE), and Virtual Organizations as Sociotechnical Systems (VOSS). See Appendix A for the full attendee list.

---

[1] 2015 Software Infrastructure for Sustained Innovation (SI^2) Principal Investigators Workshop, Frank Timmes, Stan Ahalt, Matthew Turk, Ray Idaszak, Mark Schildhauer, Richard Brower, Chris Lenhardt, and Karl Gustafson, National Science Foundation, April 2015

Based on participant feedback, the 2016 workshop achieved its five strategic objectives. In addition, the workshop expanded the collective domain of knowledge on software infrastructure, enhanced the capability for sustained innovation within software communities, and contributed to the broader CIF21 vision.

Section 1.1 – Workshop Organization and Execution

The 2016 SI^2 PI Workshop Organizing Committee held one-hour WebEx meetings once a week between October 6, 2015 and October 27, 2015. The workshop website first went online on November 5, 2015. The initial meeting with the professional facilitators, Knowinnovation, took place on November 6, 2015. The 2016 organizing committee, the 2017 organizing committee, and representatives from Knowinnovation met weekly from January 6 through February 10, 2016.

The organizing committee decided in early November 2015 that the 2016 SI^2 PI workshop would offer a traditional, top-down-driven agenda with multiple keynote speakers blended with a contemporary, on-the-fly, participant-driven agenda.

External facilitators are commonly used when it is desirable to have neutral or unbiased facilitation that can bring fresh perspectives and new questions to the discussions. The organizing committee chose Knowinnovation to provide external facilitation because the company specializes in accelerating academic, scientific, and interdisciplinary innovation. In the simplest terms, the company's facilitators help generate interesting conversations about complex questions that can lead to novel ideas and innovative research. Knowinnovation has a history of facilitating workshops for organizations that fund scientific innovation, with past clients including the National Science Foundation, National Institutes of Health, the Howard Hughes Medical Institute, and a number of United Kingdom Research Councils, including the Engineering and Physical Sciences Research Council, the Economic and Social Research Council, the National Environment Research Council, the Biotechnology and Biological Sciences Research Council, and the Science and Technology Facilities Council.

Knowinnovation staff members Tim Dunne and Costa Michailidis attended the organizing committee meeting on November 6, 2015 and attended weekly organizing committee meetings from January 6, 2016 to February 10, 2016. Based on participant feedback, the use of professional facilitators was widely viewed as a net positive added value to the workshop. Using professional facilitators allowed the SI^2 organizing committee to focus on the content of the workshop and more fully participate in the discussion sessions, instead of having to focus on the mechanics of running the workshop.

Based on the topics discussed at the 2015 workshop, NSF SI^2 PI input, and NSF Program Officer suggestions, the 2016 organizing committee decided in December 2015 to embed the five strategic objectives under four specific themes with targeted keynote speakers:
1) Training Computational Scientists, with keynote speaker Nancy Wilkins-Diehr (University of California, San Diego); 2) Software Organizational Forms, with keynote speaker James Howison (University of Texas at Austin); 3) Leveraging Industrial Software Development, with keynote speaker Laurie Williams (North Carolina State University), who was replaced by Frank

Timmes (Arizona State University) due to inclement weather; and 4) Domain-specific Software Infrastructure, with keynote speaker Shaowen Wang (University of Illinois Urbana-Champaign).

Section 1.2 Rationale for Forming the 2017 Organizing Committee

The 2016 organizing committee decided to establish a sustainable organizing committee process to provide continuity and preserve momentum between the annual workshops. A solicitation to join the 2017 organizing committee was sent to all SI^2 PIs on November 6, 2015. The following SI^2 PIs volunteered:

Amarda Shehu, George Mason University
Ganesh Gopalakrishnan, University of Utah
Jonathan Hauenstein, Notre Dame University
Kyle Niemeyer, Oregon State University
Matt Knepley, University of Chicago
Matthew Turk, University of Illinois at Urbana-Champaign
Nancy Wilkins-Diehr, University of California, San Diego
Yung-Hsiang Lu, Purdue University

The 2017 organizing committee was invited to join the weekly 2016 organizing committee meetings starting in early December 2015.

SECTION 2.0 – WORKSHOP

Section 2.1 - Format and Knowinnovation Facilitation

All workshop participants had opportunities to propose discussion topics, both before and during the workshop. Before the workshop, participants proposed 24 topics. The format for the event was a modified unconference meeting. The objective was to reduce the amount of time spent passively listening and maximize cross-project interactions that would advance the SI^2 program and individual project goals. Attendees were expected to contribute to the agenda and therefore to the success of the workshop. Participants were continually encouraged to create, lead, and actively participate in the breakout sessions. The facilitators and the participants worked together to ensure constructive outcomes. Sharing information before and during the event was critical.

The morning session on February 16 began with Frank Timmes, workshop Principal Investigator and host, welcoming the attendees and recognizing the NSF as workshop sponsor. The 2016 organizing committee members were introduced, as were the Knowinnovation facilitators, Tim Dunne and Costa Michailidis. It was explained that Knowinnovation “helps smart people have interesting conversations about complex questions,” and that this leads to novel ideas and innovative research. The four broad themes of the agenda were highlighted, followed by introductions of the keynote speakers. The participants were reminded of the participant-driven nature of the workshop and invited to choose topics of interest on-the-fly during “agenda scrums,” attend breakout sessions of their choice, and deliver breakout session reports to share outcomes with the full group. Timmes concluded the welcome talk by introducing the members of the 2017 workshop organizing committee.

Dan Katz, the outgoing NSF program officer for SI^2, introduced NSF colleague, Rajiv Ramnath, who is the new program officer for the SI^2 program. Ramnath provided an update on the SI^2 program in a presentation titled "Software Infrastructure for Sustained Innovation (SI)^2." Topics he addressed included: NSF Cyberinfrastructure Vision, Software Cyberinfrastructure Programs, Example Projects, and Future Directions. Ramnath concluded his talk by encouraging participants to think about what they need to do to make their SI^2 projects successful, and offered his help navigating the NSF to help participants achieve those goals. The morning plenary session concluded with the first 20-minute "lightning" keynote address, delivered by Nancy Wilkins-Diehr, followed by 10 minutes of questions and answers.

The facilitators then introduced the process that would be used to explore the workshop themes. This process consisted of four cycles of the following four components:

- *Keynote speech:* A speaker delivers a "lightning" keynote talk to the whole group.
- *Agenda scrum:* Participants work with other attendees at their table to generate breakout group topic ideas, then reorganize into small groups based on the specific topic they want to work on.
- *Working session:* Breakout groups explore specific topics.
- *Report back:* All groups reconvene and report back about their discussions.

To kick off the first agenda scrum, the facilitators provided each table of participants with a flip chart and markers. Participants at each table were asked to develop three topics for the breakout discussion in line with the theme of Nancy Wilkins-Diehr's keynote address. When a topic was written on a flip chart it was then placed on the wall in the hallway outside the plenary conference room. Once all the breakout topics were posted, participants were invited to exit the room, view the proposed breakout topics, and indicate which topics were of greatest interest to them using sticky dots to mark their "votes." As it became apparent which topics were of greatest interest, organizing committee members rearranged the flip chart sheets to group similar topics together. Participants then physically gathered around the topics they were most interested in, and as gatherings attained sufficient size, they were designated as breakout groups. Breakout groups were then assigned to a room, instructed to appoint a moderator and note-taker, and set up with a Google Document in which to take notes.

Throughout the workshop, the facilitators emphasized the importance of taking attendance at each breakout session and capturing notes detailed enough to lead to productive workshop reporting. It was requested that each breakout group provide a clear statement of next steps for their topic. As many as six breakout groups occurred simultaneously, and working sessions allowed breakout groups 60-90 minutes to discuss their topic before reporting back to the plenary gathering. In their reports, breakout groups were asked to provide the title of their session; the names of the moderator, note-taker and participants; a description of their issues/ motivation; suggestions for potential solutions; and a list of next steps. Once the participants returned to the plenary room, the facilitators identified a spokesperson for each breakout group. The Google Document containing notes for each breakout group was projected during the group report to provide a reference for the spokesperson and plenary participants as each summary was presented. The same process was followed for each of the two subsequent workshop themes that were explored on the workshop's first day.

The evening of the first day concluded with a two-hour poster session. Attendees were required to submit digital copies of their posters in advance and bring a printout to display. The poster session enabled participants to interact and collaborate. The facilitators prepared a “poster session feedback area” in the poster session venue, where participants were invited to post notes describing new collaborations, interesting posters, and other feedback. In a show of enthusiastic support, participant Ilya Zaslavsky took all the electronic versions of the posters and entered them into an online visualization and analysis system called "Survey Analysis via Visual Exploration" (SuAVE). SuAVE is an one of the outcomes of NSF ACI-1443082 “EAGER: Development of a Novel Online Visual Survey Data Analysis Tool and Assessment of its Capabilities to Enhance Learning of Quantitative Research Methods” project. It provides a framework for online sharing of surveys and visual materials, and offers an alternative, elegant, and visually-oriented navigation scheme [See Zaslavsky,I. http://maxim.ucsd.edu/suave/main.html?file=si2n.cxml&views=111000]

The workshop’s second day opened with a short speech in which Dan Katz encouraged participants to avoid phrasing outcomes or goals as “what do you want the NSF or others to do,” and instead focus on “how are we going to get the NSF and others to achieve the next steps and changes we've identified.” “You, the SI^2 PIs, are the reviewers, proposers, and community,” Katz said.

The facilitators then guided the final two scrum agenda cycles, using a slightly different process for the sake of efficiency and variety. Instead of using the flip charts and voting system, the facilitators invited participants to verbally pitch breakout topics to the larger group, which then voted to indicate their level of interest in the topic. Another key difference was that in the fourth thematic agenda scrum (the first one held on the workshop’s second day), in addition to topics related to the keynote theme, participants were also invited to pitch topics of interest that were off-theme. This breakout session, which yielded four topics, was titled “Soap Box.” Following the breakout sessions, participants reassembled in the plenary meeting space final report outs, in which the facilitators efficiently managed timing and provided microphones for all speakers. The same revised scrum process was employed for the final breakout sessions, which addressed the theme of “Next Steps.”

Near the conclusion of the workshop, Frank Timmes spoke on behalf of the workshop participants to thank Dan Katz for his service to the community and for his contributions to the NSF SI^2 program.

The workshop concluded at 12:00 p.m. on Wednesday, February 17. An initial draft of the workshop report was created March 1, 2016, made available to workshop attendees for comments on April 11, 2016, and submitted to the NSF on April 15, 2016 for public dissemination.

Section 2.2 - Keynote Speakers and Topics

Keynote #1

Theme: Training Computational Scientists
Talk Title: Cyberinfrastructure Uptake: Reflections over 20 years
Speaker: Nancy Wilkins-Diehr

Keynote #2

Theme: Software Organizational Forms
Talk Title: Open Source and Scientific Software Production: Different ways of getting the work done
Speaker: James Howison

Keynote #3

Theme: Leveraging Industrial Software Development
Talk Title: Modules for Experiments in Stellar Astrophysics
Speaker: Francis Timmes

Keynote #4

Theme: Domain-specific Software Infrastructure
Talk Title: Open CyberGIS Software for Geospatial Research and Education in the Big Data Era
Speaker: Shaowen Wang

Section 2.3 - Agenda

Tuesday, February 16	
7:00 a.m.	Continental Breakfast
8:00 a.m.	Opening: Knowinnovation Welcome Address: Frank Timmes
8:15 a.m.	NSF SI^2 Program Dan Katz and Rajiv Ramnath
9:00 a.m.	Keynote Address #1 Theme: Training Computational Scientists Talk Title: Cyberinfrastructure Uptake: Reflections over 20 years Speaker: Nancy Wilkins-Diehr
9:30 a.m.	Agenda Scrum
10:00 a.m.	Coffee Break
10:15 a.m.	Breakout Sessions
11:30 a.m.	Breakout Session Reports

12:00 p.m.	Lunch
1:00 p.m.	Keynote Address #2 Theme: Software Organizational Forms Talk Title: Open Source and Scientific Software Production: Different ways of getting the work done Speaker: James Howison
1:30 p.m.	Agenda Scrum
2:00 p.m.	Breakout Sessions
3:00 p.m.	Coffee Break
3:15 p.m.	Breakout Session Reports
3:45 p.m.	Keynote Address #3 Theme: Leveraging Industrial Software Development Talk Title: Modules for Experiments in Astrophysics Speaker: Frank Timmes
4:15 p.m.	Agenda Scrum
4:30 p.m.	Breakout Sessions
5:30 p.m.	Breakout Session Reports
5:45 p.m.	Poster Setup
6:00 p.m.	Poster Session
Wednesday, February 17	
7:00 a.m.	Continental Breakfast
8:00 a.m.	Keynote Address #4 Theme: Domain-specific Software Infrastructure Talk Title: Open CyberGIS Software for Geospatial Research and Education in the Big Data Era Speaker: Shaowen Wang
8:30 a.m.	Agenda Scrum
8:45 a.m.	Breakout Sessions
10:00 a.m.	Coffee Break
10:15 a.m.	Breakout Session Reports
10:30 a.m.	Next Steps Agenda Scrum
10:45 a.m.	Next Steps Breakouts
11:45 a.m.	Next Steps Reports
12:00 p.m.	Box Lunches Workshop Adjourned

SECTION 3.0 - BREAKOUT SESSIONS

Below are summaries of the discussions and outcomes of the 26 breakout sessions held during the workshop. More detailed notes on each session can be accessed by visiting the Google Documents linked to each section or breakout group title.

Section 3.1 - Breakout Session #1: Training Computational Scientists

For the theme "Training Computational Scientists," keynote speaker Nancy Wilkins-Diehr (San Diego Supercomputer Center, University of California, San Diego) delivered a presentation titled "Cyberinfrastructure Uptake: Some observations over 20 years." Following the presentation, six participant-driven breakout sessions were held on topics surrounding the theme.

3.1.1 Training Students

Participants: Abani Patra, Ganesh Gopalakrishnan, Robert van de Geijn, Gagan Agrawal, Ganesh Gopalakrishnan, Abani Patra, Kyle Niemeyer, A. Selcuk Uluagac, Geof Sawaya, Margo Seltzer, Inanc Senocak, DK Panda, Anthony Aufdenkampe

Issues/Motivation:

- Fundamental new way of doing science (computation underlies everything)
- Pedagogy has not kept up

Challenges:

- Awareness of the sea-change in the way science is being done has not been translated into courses downstream
    - Related: institutional/department inertia, old ways of thinking/rigid disciplinary boundaries
- How to fit it all within the allotted curricular credits
- What domain scientists need is typically not what computer science courses teach

Potential Solutions:

- Will the exemplar (IEEE TCPP book on par computing) be a model for promoting awareness?
- Can we inject small doses of programming (beyond Matlab) in many classes?
- Suggestion to teach fundamental abstractions in the domain, and the fact that these can help modularize problem-solving
- Code.org is indicative of sea-change coming in terms of code awareness
- Can build on momentum in other similar areas (e.g., Python)
- A potential NSF-funded workshop on this topic could focus on how to revamp undergraduate education to serve domain science needs. Running it under the auspices of the Computing Research Association, the Computing Community Consortium, the Society for Industrial and Applied Mathematics or a similar organization would help to sustain such an endeavor.
- SI^2 should "own" (or be responsible for) incentivizing educational activities in this area

- We don't want "layers" that can't be penetrated (need to create cross-layer awareness and how things work under the hood)
- Need to bridge between "domain science" (in the computational sense) and "social science" needs (graph problems, linear algebra, etc.)
- Quality of delivery is important (domain experts offering computer science to domain science folks)
- Those who evolve curricula need to be facilitators with regard to teaching

3.1.2 Student Projects: How to Improve Quality of Student Code

Participants: Ilja Siepmann, Yung-Hsiang Lu, Paul Butler, Senatham Writn, Paul Navrafil, Greg Tuckor, Ilja Siepmann, Glenn Martyna, Thomas Cheatham, Ross Walker, Cyrus UMrigor, Neil Hearman, H. Birali Runesha, Hazeline Asuncion

Issues/Motivation:

- How to help students start new projects in a good way?
- How to manage projects that have been growing for years?
- How to facilitate collaboration between computer people and domain experts?

Potential Solutions:

- Create a bootcamp as an SI^2 community collaboration to teach basic best practices (potentially with NSF support). Example: project at the University of Chicago run by Hakizumwami Birali Runesha.
- Create models for students/scientists to get credit for the code written by students. Motivate students to write better code so that they can be recognized.
- Create a software engineering course taken by graduate students, certificate for software engineering
- Create better communication software tools, such as Slack
- Help students understand commercial software development so that they develop better practices
- Study failed projects and understand the problems
- Create online tutorials for students to learn what is the best practice (may need community effort)
- Train students to do something (thinking, designing) other than/before writing code
- Develop methodology for balancing the conflicting goals of research outputs, getting data that can be used in papers, and meeting the needs of sponsors
- Revise evaluation criteria to include end-user value-add metrics, such as: How many use the software? How many people find it useful in solving their problems?
- Commercialize the software

Next Steps:

- NSF could create a program and solicit proposals (or expand from existing grants) to create short courses (a few days or a few weeks) on best practices
- Collect information from different universities about their approaches in training software engineering as part of NSF education
- Difference between computer science and software engineering: Treat software engineering as a domain

- People need to know software tools, but how much knowledge is needed?
- Research projects grow gradually, without a clear plan at the beginning
- Can a project be re-implemented after the research project has demonstrated the scientific value? Should this be done by the same student? Can this be done by another student?
- Change the habit from short-term homework assignments to long-term research projects
- Do universities teach software engineering to all (or most) students?
- There are a lot of tutorial videos online, but students may not know which one to watch

3.1.3 Models for Sustaining Software

Participants: Ed Maginn, John Mellor-Crummey, Ed Magin, Jason Leigh, Steve Siegel, Gene Cooperman, Bruce Berriman, Alberto Passalacqua, Marlon Pierce, Kyle Chard, Ilya Zaslavsky

Issues/Motivation:

- Transitioning from software developed and supported by a small group to community infrastructure takes considerable effort
- What best practices can be employed to make projects successful and sustainable?
- Scientific domain software does not have the same potential for wide user base
- Is licensing a factor for success?
    - GPL (Copyleft) versus more permissive licenses, e.g., Apache

Models:

- Main developers all know each other personally
- Linux kernel model
    - Open source, but tightly controlled
- Apache model
    - Provide a social network of people who mentor projects: Releases, licensing, adding contributors, making design decisions collectively
- LLVM
    - Apple as a champion
- NAMD
    - NSF-supported molecular dynamics code: Key is large user base
- AstroPy
    - Community Python library for astronomy
- Open source versions of Matlab
    - Scilab - Scilab Enterprises maintains the code; funding model is consulting and fee for service
    - Octave - active development has slowed
- OpenFOAM
    - Free open-source code; highly modular
    - Paid training and consultancy/development
    - Highly-selective acceptance of code contributions from the user community
- Unidata

Potential Solutions:

- Quality of design is important

    - Enables new contributors to focus on a small part and add something
    - LLVM example
- Plug-in models are successful for attracting new contributors
- Google summer of code to aid incubation of projects
    - Support Apache projects
- Sustainable projects have someone that keeps funding stream coming in
- Industrial sponsored development of particular features
- Federal funding to sustain a project until it achieves critical mass
- Federal center to support open-source software
    - Distributing funding to groups
    - Providing technical support for projects (software engineers)
- Subscription model for support

Next Steps:

- Encouraging universities to contribute support for research software?
    - Can doing so provide a competitive advantage?
- Encourage communities
    - Provide funding to multiple groups with requirements for community collaboration
- Explore funding opportunities beyond NSF
    - Other federal agencies
    - Consortia of universities
        - NSF-funded institutes
    - Industrial-supported consortia
- Explore whether a federal center is a feasible approach
    - Governance: selection, continuation, sunsetting
    - What would the center provide?
        - Funds
        - Software engineers to contribute

3.1.4 Interdisciplinary Collaboration

Participants: Zach Byerly, Steve Brandt, Gerry Puckett, Tom Jordan, Tom Miller, Theresa Head-Gordon, Mike Kirby, Robert J. Harrison, David Tarboton, Richard Brower, Jose Fortes, Peter Elmer, Nancy Wilkins-Diehr, Chris Rapier, Xilun Chen, Reuben Budiardja, Saday Sadayappan, Emilio Gallicchio, Ewa Deelman, David Schloen

Issues/Motivation:

- Is it creating an interdisciplinary project, or is it a new discipline?
- One perspective on generating interdisciplinary people: Train them up in their discipline. Teach them to: 1) communicate (be multi-lingual scientifically); 2) respect (appreciate that they must actively work to respect their collaborator's area and vice versa); and 3) remain competent in their areas (a collaborator will assume you are remaining core to your area).
- What kind of career do you have if you have multiple disciplines or change disciplines?
- In interdisciplinary collaboration, all groups need to respect and learn about each other, and one should not feel as if it's being subjugated

- Need different kinds of people: 1) people strong in their disciplines, who stay in their disciplines, and 2) a facilitator, a person who works with both and communicates and provides coordination to make things work.
    - Question: Is facilitation a skill that can be trained/added on top of disciplinary training, or does it have to be integrated in a person's training from the start?
- Hiring a facilitator with no grant funding is a non-starter, but once a grant is present, they may be fought over
- Need a name and position for these facilitators
- Joint appointments (those tenured in two departments) are part of this
- Is there a tenure track for a computational scientist? Perhaps the label is too broad?
    - Research software engineer
    - In U Utah, the School of Computing is bigger than Computer Science
- Real challenge to a career in computational science is the time it takes to obtain results. It could be 50 years. What does it take to make that gap smaller? How can we get people up to speed in computational science more quickly?
- Motivation for computational scientists and disciplines are driven by the needs of the discipline and the market. This is not an easy problem to solve.
- Need a classical mathematical background to be a good computational scientist
- How to create domain scientists and give them the computational background they need? What is it that we do right now?
- How much of the challenge is labor intensive? Can we have an abstract layer for scientists to work on? How much of the problem is a lack of tools?
- Is it better to come from domain science or computer science? Both paths have worked. One model that used at the University of Pennsylvania is to take undergraduates that have been trained in math and physical sciences, and then add on a layer of training (Master of Software Design or similar) that provides the essential components for interdisciplinary work, such as coding, working in teams, design, etc.
- Some disciplines have bodies of knowledge, what people should know to practice. University of Utah is exploring this, University of Pennsylvania has a program like this, an 18-month program to turn a math-heavy degree into a master's in computational science. Indiana also has a good school of Informatics (network infrastructure, cyberinfrastructure, etc.). San Diego has good efforts in bioinformatics. Michigan State has started a department of Computational Mathematics.
- Profile of a computational scientist has changed dramatically over the last three years. Now they need to understand databases, machine learning, data mining, etc.
- Schools of Informatics that have popped up may be closer to what a computational scientist needs to be
- Companies recognize computer talent better than universities
- Set up a lab with a new name, if they can put them into other departments
- No place in the application where you document your computational skills. How do we evaluate people's ability to be computational scientists?

Potential Solutions:

- Introduce computer science and domain scientists at an earlier stage, e.g., bringing (interested!) computer science undergraduates into domain science research via Research Experiences for Undergraduates grants
- Creating a rapid training of domain scientists in a computational science area

- Example: The Integrative Graduate Education and Research Traineeship brought students together and they trained together
- Re-thinking computational science/informatics
- Main challenge is the time to publish can be fixed if it is systemic (funding, education)
- Encourage universities to support research staff (another group talked about how to do that)

Next Steps:
- Encourage NSF to fund the types of programs mentioned above for individuals pursuing these careers

3.1.5 Community Building

Participants: Jerry Bernholc, Rich Vuduc, Peter Cummings, Andreas Stathopoulos, Nicholas Gottschlich, Matthew Turk, Matthew Newville, John Clyne, Coray Colina, Benjamin Winjum, Jerry Bernholc, Rene Gassmoeller, Shava Smallen, Ken Subratie, Ashish Gehani, Dmitry Pekurovsky, Stacy Lynn, Mike Hucka, Robert McLay

Issues/Motivation:
- There are a lot of people developing tools that they would like others to be able to use. How do they advertise what they have? How do they build a user community? How do they sustain such a community?
- If you develop solutions, you want to learn about new developments and tools that you can integrate into your system. So, developers also want to keep on top of what other developers are doing. This is not just about developers reaching end users, but also you learning from other developers, getting their input, getting their contributions.

Challenges:
- When trying to build a community, a big challenge (and perhaps the first) is figuring out what you want from the community
- Getting “active users” is more difficult than getting users. Using the software is not necessarily being part of the “active community” of a particular piece of software. How do you do it?
    - People may need further motivation: “How does it benefit *me* to become part of an active community?”
- Getting feedback from users is difficult. Filling out survey forms is not popular.
    - Acting on the feedback is also difficult. Some requests may be very difficult to implement. How do you choose which ones to address/implement?
- Example case: NanoHub
    - With 3-4 attendees in nanoscience, most attendees were not very aware of NanoHub
    - One person reports that an industry group attempted to use NanoHub for a small-scale research problem. They tried to use classical simulation, failed, came back and tried a different simulation. This may not be the typical experience, however.
    - What does NanoHub do in terms of facilitating work and connections between researchers? What are the features and capabilities that support user

communities?

Potential Solutions:

- How do you reach out to users?
  - Contacting colleagues individually
  - Going to meetings: presenting your work and talking to people. Explaining to people how you have something that will let them do their work better or more efficiently.
  - Publications
  - Hubs (like NanoHub)
- How do you find resources and software?
  - Search the web (Google)
  - Look in publications
  - Ask people at meetings
- Tap into “power users” to get more detailed feedback
  - It can be desirable to turn power users into project contributors
  - As software developer, you need to think about the different barriers to usage on the one hand, and software contributions to your project
- Multiple mechanisms for communicating with users and engaging new users:
  - Using GitHub and SourceForge, you can reach people, but it may be more oriented towards users. However, end users do use these for downloading software. At least a couple of projects in the room have done this, and find their users download from SourceForge.
  - Mailing lists remain a common approach for communication and feedback, but people are finding increasingly that questions are posted by users via issue trackers and other mechanisms
  - Other, newer approaches: Twitter, Facebook, Slack
  - Seeing feedback being taken seriously is important to potential end users; as a result, it is helpful to make the mailing lists and other feedback mechanisms visible (e.g., no-wall, public posting)
  - Internet Relay Chat is another approach that may be popular in some communities but seems to be on the wane
- Sneaky approaches:
  - Have people post questions on social sites deliberately (for example, an undergraduate could be asked to post something each week)
  - Get other people to answer questions, not just the core developer(s)
- What do users get out of answering questions and participating in discussions? What are their motivations?
  - Human urge to help
  - Meet other people who may have similar interests and thus may be people you can turn to for help or cooperation in the future
  - Building an online reputation
  - There’s also research on this (e.g., about the Apache help forum)
    - People learn things by answering questions
- How can you convert active users into developers/contributors?
  - One approach: hold workshops, show people how to work on the code itself

    - Pick a language or framework that people want to use or are familiar with (e.g., Python versus Fortran)
    - Write another grant proposal to develop new features and include the people on the grant
    - Hold regular webinars, on different topics: training, user-driven questions. Also helps prioritize new feature development.
    - Tools for large numbers: Adobe Connect, GoToMeeting
        - An institute could negotiate rates for commercial systems
    - Hold a “Feature Friday” online presentation about new things in your software, or to provide answers to questions
    - Provide skeleton codes to help developers (or advanced users) get started
    - Provide input cases, test cases, or sample problems for them to try out to learn how to use the software. Professors can use these in classwork to illustrate topics, helping to further disseminate them.
    - Pick a good license

Additional resources:

- Book: “Buzzing Communities” (Richard Millington)
- Guide: “How to Ask Questions the Smart Way” (Eric Raymond)
- Book: “Producing Open Source Software” (Karl Fogel)
- Blog post: “What makes a good community?” (Sarah Sharp)
- Book: “Building Successful Online Communities" (Robert E. Kraut and Paul Resnick)

3.1.6 Career Incentives for Computer Science and Engineering Research

Participants: Simon Hettrick, Shawn McKee, Ben Haley, Gregg Musiker, Sameer Shende

Issues/Motivation:

- How do we sustain research capabilities developed as part of projects for broader, continuing use?
- How do we create/motivate universities to create long-term or tenured positions bridging domain science, research software engineering, and research infrastructure engineering?
- Is publication the right route to emphasize? Important to recognize the value of research development and support work.
- What kind of positions exist at universities that could be repurposed or adapted to create desired positions?
- Problem: “IT” is where these people get sent. Need a more appropriate center/location. What we are discussing is not “IT” as it is usually instantiated.
- Need to differentiate the requirements for this new “thing” the universities might create for research software/infrastructure engineers versus existing “researchers” within domains
- Currently always comes down to “soft money”
- Universities are slow to change. Need to demonstrate the importance (based on existing efforts) so they are motivated to explore solutions.

Potential Solutions:

- Can we increase the acceptance of technical publications as having “weight” in academia?

- Encourage (strongly) citing the use of software and infrastructure to build credit for those involved in creating, developing, and maintaining those things.
- Highlight the roles of those bridging the domain science and research infrastructure (software and hardware) in the science process. The goal is to lead universities to develop suitable permanent positions to allow the right environment to grow.

Next Steps:
- Solicit creation of research centers?
- Define the requirements and evaluation criteria for those who bridge science domains and research infrastructure (need to differentiate with existing researchers)

Section 3.2 - Breakout Session #2: Software Organizational Forms

For the theme "Software Organizational Forms," keynote speaker James Howison (University of Texas at Austin) delivered a presentation titled "Software Ecosystems: Spreading the work around." Following the presentation, six participant-driven breakout sessions were held on topics surrounding the theme.

3.2.1 Small Teams

Participants: John Clyne, Matthew Newville, Benjamin Winjum, Andrew Christlieb, Gary Olson, Tom Jordan, Bruce Berriman, Gregg Musiker, Mike Hucka, Emilio Gallicchio, Jason Leigh, Alberto Passalacqua, Robert McLay, Anthony Danalis, Ganesh Gopalakrishnan, Rion Dooley

Issues/Motivation:
- How can small teams (the definition of which depends on the organization) manage "non-coding activities" associated with a project?
- While best-practices such as 'continuous integration' are recommended, tests often cannot run (in a continuous integration sense)
- Differences of small groups versus startups
    - Leadership model
    - Continuously changing workforces
    - Acceptable to have delayed payoff
    - Obligations with regard to funding agency are different
- Key issues, in order of priority:
    - Communicate vision/roadmap
    - Get funding
    - Establish and maintain communication channels, tutorials, documentation, and examples of usage
    - Attract outside developers
    - Consulting, developing unit-tests, in-person training, bug-tracking, and resolution
- Importance of creating and sustaining documentation (especially API) and tutorials:
    - Mailing lists and forums
        - Difference between "light" and "heavy" amount of investment in time
        - Webinars
        - On-site visits
        - Continuous integration

        - Unit testing
        - Building examples
        - Sample data
        - Birds of a Feather Meetings at conferences (just not over the lunch break)
        - Providing public roadmap
        - Keeping the code running (quick bug fixes retain community)
        - Kinds of users
- Helpful users versus users that are not conducive to advancing the software
- Communicating roadmap, vision
- Role of community in doing this (funding agency may have a say in this)
- Funding to keep things going
    - Question: Need to plan for growth (each addition of a project member requires expending training time)
- Discussions centered on risk
    - Team size versus stomach for risk
    - Mitigating risk, and contingency (in terms of pubs not happening without backup plans)
- Upstream/downstream communication:
    - Different for small teams

Potential Solutions:

- Keep continuous integration going, and respond to it quickly (to avoid being de-sensitized to error messages)
- Small teams cannot expend expensive resources (PI time, for example) in doing all tasks (e.g., mailing lists). Community ought to recognize value-proposition and step up.
- Need a “benevolent dictator” model
- Finding other partners, even commercial ones, or do a startup, having someone else take it over (e.g., one’s university, a national lab)
    - Issue of possibly giving up ownership
- Extended Collaborative Support Service (ECSS)-like support:
    - Can help small teams, but comes with associated obligations in terms of how to best present one’s questions and get the best help, apportion credits correctly
- Pilot approaches (when funded) can make it cheaper to fail, thus mitigating risk
- Going away from a contract model (ensuring a redeeming value and perpetuating further enhancements) can help science

Next Steps:

- How can the SSE/SI^2 program help?
    - Small groups: how to foster and facilitate their efforts (funding, programs)

Resources:

- Startup experience links:
    - SXSW Interactive: http://www.sxsw.com/interactive
    - Alexander Muse blog: http://startupmuse.com/
    - The Startup Bible: http://amzn.to/1OeO9na
    - The Lean Startup (Eric Ries): http://www.kimhartman.se/wp-content/uploads/2013/10/the-lean-startup-summary.pdf

3.2.2 Rewards and Incentives for Openness (and addressing costs)

Participants: David Tarboton, Ed Maginn, Ilja Siepmann, James Howison, Kyle Chard, David Schloen, Gerry Puckett, Rene Gassmoeller, Janos Sallai, Amarda Shehu, Stacy Lynn

Issues/Motivation:

- How do we encourage people to contribute who are not computer scientists and do not have an innate drive to do open source?
    - Why do people need incentives? Mode of funding, publications
    - Looking at the "stream of publications," it seems clear that original authors are getting co-authorships from holding the code tight. Seems hard to replace that possibility; perhaps if there is scope to have many more publications using the techniques, but the original authors getting citations, rather than co-authorships.
- If you participate in helping develop code, how much help warrants authorship?
- There is overhead in sharing code; how do you lower this overhead to make it easier for people to open their codes?
- Valley of death: Initially there is a lot of work, and the rewards come later. It is difficult to do this when there is a small community. How do you minimize the pain?
- Promoting your code and advertising at conferences is important to shorten the transition time from release to widespread adoption
- Reproducibility/validation of scientific results
- Tension between having a student develop new software that doesn't result in a publication versus the student running existing code to do new science and getting a publication
- Barrier to open source: feeling of worry that others will find bugs in the code. Potential for being embarrassed by code.
- Work on code in open source establishes a record that can advance careers and get jobs. (This is more than anecdotal; see Marlow, J., & Dabbish, L. (2013). Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 conference on Computer supported cooperative work* (pp. 145–156). New York, NY, USA: ACM. http://doi.org/10.1145/2441776.2441794)
- Is there a fear that by going to a community code, you end up with the least common denominator? Or do community codes give you an aggregation of all the best ideas?
    - Monolithic software requires agreement, which can stifle innovation
    - Development slows down when you need to organize everything
    - The key is to have a design that is modular and not so interdependent
- What is the business model of open source? The price is $0, so what are the other rewards?
    - Reputation is key in academia
    - Citations

Potential Solutions:

- Provide outside resources that can provide the infrastructure to help launch codes and develop the infrastructure, for example the NSF Institute: Computational Infrastructure for Geodynamics (CIG)
    - http://www.nsf.gov/awardsearch/showAward?AWD_ID=0949446

    - http://www.nsf.gov/awardsearch/showAward?AWD_ID=0426271
    - http://www.nsf.gov/mobile/news/news_summ.jsp?cntn_id=100442&org=NSF
- Get a digital object identifier (DOI) for a specific version of your code (for example, through zenodo.org), which a student can use as a way of gaining recognition for code development that might not show up in a publication
- Having letters of recommendation that point out strong coding ability can be helpful for students. Also, a strong presence on GitHub has value to employers. So the key is being able to publicize what students have done with a code. It is not all just papers.
- Universities should recognize software as part of faculty merit raises
- Besides citations to a paper reporting the software, there should be other ways of recognizing software impact, such as downloads
- For every product, develop a map of contributors (for example, using Project CRediT): motivate the idea for how to record credit on different research artifacts. (For further discussion, see this presentation given by Dan Katz.)
- Ongoing use with your own tools. (Century Soil Organic Model as an example)

Next Steps:

- Have or find materials that can articulate the benefits or rewards of contributing to your particular open source project
- Influence scientific journals and editors to set policies on what should be cited or recognized. Encourage them to come up with ways of recognizing software contributions.
    - When publishing a paper, identify all codes (with version numbers) used
    - Create credit maps for community codes, which can be used to identify who contributed what
- When open source code developers extend their codes to help users solve new problems, consider making the code developers co-authors

3.2.3 Passive to Active Openness (community)

Participants, Robert van de Geijn, Abani Patra, Paul Navratil, A. Selcuk Uluagac, Ken Subratie, Steven Brandt, D.K. Panda, Jerry Bernholc, Nicholas Gottschlich, Andreas Stathopoulos

Issues/Motivation:

- What is the definition of passive versus active openness? The difference between the "consuming" versus "contributing" community is one aspect. Sensing upstream and downstream.
- What is the role of the gatekeeper (which could be a subset of the community, or an individual)? An important role is quality assurance.
- How does one balance standards and sustainability versus ease of contribution? Modularity for contribution or rigid structure as endpoints in a spectrum?
- When does a person become an active contributor? When one submits a bug report, for example, or only when contributing code?
- How to train contributors?
- What mechanisms are needed for establishing provenance and rights to code?

Potential Solutions:

- Test suites can be instrumental in enabling/promoting active open communities
- Pedagogical "sandbox" can be used to encourage people to look (and contribute) "under the hood"
- The "bazaar" (open view) model is probably better than "cathedral" (high-priest) model
- Active open communities are supported when there's a mechanism for communicating your changes and roadmap to allow others to prepare, for example, through membership in other software that is closely related (e.g., upstream or downstream from what yours is doing)
- Automation, leveraging tinker toy when possible
- Formal bug reporting (ticketing of bugs, community response policy)
- Educate the community on intellectual property issues

3.2.4 Instrumenting-Monitoring-Codes versus Privacy

Participants: Shantenu Jha, Ross Walker, Thomas Cheatham, Peter Elmer, Gene Cooperman, Margo Seltzer

Issues/Motivation:

- Obvious way to gauge users is via code instrumentation; however, when does this cross bounds of privacy?
- Do not want to focus on privacy issues—we are not lawyers.
- Real point is that some instrumenting could be extremely useful. Clearly moving or reporting back raw user data is bad, whereas aggregated data is probably OK; however any data movement needs strong technical justification.
- Different levels of instrumenting/reporting are possible. We discussed levels ranging from monitoring the build/configure versus runtime in application stack (internal actions

on measured data versus phone home/invasion of privacy but could be used to improve performance/usage) versus external monitoring (Xalt).
- What can sensing/instrumenting/monitoring give you? Hopefully information on what can make your code/application better, faster, more optimized, more power-efficient?
    - Impact/benefit (statistics) for reports (save time by avoiding user surveys)
    - Capturing build failures on specific platforms/packaging errors/configurations in use; Notice of use of older APIs or poor/outdated code paths
    - Proposing updates on-the-fly based on usage
    - Record use cases (where to focus development effort)
    - Detect duplicate efforts (code paths run by others, borders on privacy violation)
    - Performance people are obtaining
    - Real-time code/binary/hardware check (i.e., determinism on GPU can look for deviations in short test run report code or hardware error) enables more detailed crash tracebacks for debugging
- We explored cost-benefit questions surrounding pushing the bounds on user privacy. When might it be beneficial to approach that boundary?

Potential Solutions:
- Application developer ethical check: If a user calls you up (found code snippet) could you justify the instrumentation need and appropriate use and/or would you be embarrassed? What data can we get access to? Whatever the user allows.
- Need clear disclosure of data access, aggregation, and provenance that the application monitors.

Next Steps:
- Unclear. Would be very useful to know how many codes do auto-reporting back.
- Straw poll: Eight attendees claimed code monitoring/instrumentation capabilities are in active use on their projects.

3.2.5 Community and Innovation

Participants: Paul Butler, Nancy Wilkins-Diehr, Marlon Pierce, Shava Smallen, Reuben Budiardja, Geof Sawaya

Issues/Motivation: “When does community get in the way of innovation?”
- Breakages can be disruptive to workflows and users
- How can innovations that require breakages be implemented?
- Impedance mismatches between innovation and stability
- Funding mismatch between sustained development and supporting new innovation
- “I’ll just re-invent”

Potential Solutions:
- Cost benefit analysis in detail
- Incremental development versus disruptive changes
    - Encourage all starting projects to include large fraction of effort on continual redesign
    - Assume you will make a major overhaul

- Tools to help transition between compatibility levels
- Semantic versioning (http://semver.org/)
    - Continuous delivery

Next Steps:

- Proposal evaluation could include whether sufficient resources are being requested for stewardship of code base long term
- Encourage upstream and downstream sensing
- Encourage careful estimation of cost/benefit of disruption
- Manage expectations
    - Roadmap
    - Community engagement, communication

3.2.6 Best Practices for Evolving Software

Participants: Mike Kirby, Yung-Hsiang Lu, Inanc Senocak, Adam Updegrove, Shawn McKee, Chris Rapier, Stephen Siegel, John Mellor-Crummey, Dmitry Pekurovsky, Zach Byerly, Birali Runesha, Robert Harrison, Ashish Gehani, Jose Fortes, Sameer Shende, Coray Colina

Issues/Motivation:

- What are the best practices for evolving software, hardware, and scientific needs?
- How to balance stability and cutting-edge innovation?
    - Little fund for doing things (software) better after better understanding of the scientific process. Little fund encourages innovation, keeping up with scientific discovery and hardware improvement.
    - Protect efforts and prevent competitors from using the software for their projects
    - How to sustain software after the project's funding has ended?
    - Some grants update hardware but few grants refresh software
    - Unpredictable funding for support maintenance of software
    - Conflicting goals among students about innovation, research, publication, and software implementation and maintenance
    - Sometimes they don't know the need and sometimes they know the need but they don't know what to do
    - Good bug tracking system
    - Enforcement may be counterproductive

Potential Solutions:

- A workshop/working group to create "best practices" that can be agreed on by the community to address issues such as version control, issue tracking, automatic testing, continuous integration, and documentation
- NSF (and other funding agencies) highlight and promote groups that have best practices in software management
- NSF provide supplemental funding for "hardening software" (perhaps Research Experience for Undergraduates grants?)
- Grant review panelists pay more attention to the data management (especially software management)
- Proposals should list needed software (and acknowledge the contributions)

- Need a model to recognize the software developed by other groups
- Allocate funding for maintaining software (after the main discovery challenges are over)
- Communicate with students and researchers about best practices and how to implement them
- Create “responsible conduct of software development” (equivalent of “responsible conduct of research” for research related to human subjects)
- Examples showing good and bad things so that people know what to do and what to avoid
- Create a community to agree on best practices and enforce these through requirements by multiple federal agencies
- NSF (and other funding agencies) could require a “software management plan”
- A development platform provided by sponsors

Next Steps:

- Create a project that shows the loss of productivity due to bad practices
- Evaluate the suggestions from the 2015 report?

Section 3.3 - **Breakout Session #3: Industrial Software Development**

For the theme “Leveraging Industrial Software Development,” keynote speaker Frank Timmes (Arizona State University) delivered a presentation titled “Modules for Experiments in Stellar Astrophysics (MESA).” Following the presentation, five participant-driven breakout sessions were held on topics surrounding the theme.

3.3.1 Professional Productivity

Participants: Anthony Aufdenkampe, Stacy Lynn, David Tarboton, Mike Hucka

Issues/Motivation:

- How to get productive collaboration between:
  - Academic coders
  - Professional software development firms
  - User community (beyond the academic community)
- A collaboration between academic coders and a professional software firm requires funding. Funding models:
  - Co-funding
  - Subcontract
  - Piecemeal
- Attributes of professional software developers
  - Cost $150-$250/hour (fully loaded)
  - Very productive per hour
  - Understand professional practices for modularity, version control, issue tracking, testing, etc.
  - Don’t understand end-user
- Attributes of academics

    - Develop the scientific algorithms
    - Don't understand professional practices
    - Understand the end-user's needs
- Need to effectively communicate your vision
- Fitting cost of professional software developers into the constraints of grants and granting agencies

Potential Solutions:

- Approaches that work often include a clear division of duties. For example, have a collaboration protocol, and:
    - Hire one professional to guide academic coders through learning professional practices, such as code reviews (and QA/QC in general, which academics may not be aware of or expert in), to work as part of the team
    - Hire firm to lead development, with algorithm development by scientists
- There seems to be a strong need for a middle person who understands the end-user need and "speaks multiple languages" to facilitate exchanges between software developers and academics
- Bringing the extended working team together in one place at the onset of work to frame goals and expectations (and get to know each other) helps with successful outcomes
- Hold workshops to inform about new software, new platforms, new technologies to keep current with new tools and frameworks for development. For example, platforms such as GitHub help participants to keep current (Side note: GitHub keeps coming up at this meeting; there is an evident need to learn how to use it to its greatest benefit and for more sophisticated applications).
- Google Summer of Code internships offers an interesting model for student training. It involves a competitive application process and students work on open source projects from where they are.

Next Steps:

- Need to make partnerships with academia more attractive to industry
- Companies occasionally teach courses, which then gives them a pipeline of qualified students to hire who are experienced with industry standards. How this is done depends on mutual interest and areas of specialization.
- Lower the barrier for bringing people into the system for hire (such as in GitHub)
- Develop shadow/sandbox projects
- Iterative and agile methods
- Google Summer of Code model
- Garner more support from funding agencies and improve proposal reviewers' understanding of the true cost of supporting professional software developers

3.3.2 Supporting Next-Generation Hardware

Participants: Ross Walker, Emilio Gallicchio, Tom Cheatham, Robert van de Geijn, Matt Anderson, Bruce Berriman, Tom Jordan, Rich Vuduc, Dmitry Pekurovsky

Issues/Motivation:

- How to make code compatible for next-generation hardware platforms?

- High-performance computing versus grid computing versus desktop computing (and even smaller devices, such as iPads?)
- What is next-generation hardware? Can we predict what the next generation of software will be in order to support it?
- Competition is good for innovation but might duplicate efforts
- When new hardware comes out, we have the expertise (and desire) to port code to take advantage of it. The problem is that each time we have to start over. As human beings we have to remap our problem to the hardware. Is there a way to translate the human understanding of the algorithmic aspects of the problem to the hardware at a high level of abstraction?
- Heterogeneous hardware

Potential Solutions:

- Partnership between Department of Energy (DOE), potentially Department of Defense, and NSF and other agencies should be explored in the context of the National Strategic Computing Initiative. Example of DOE-supported co-design centers to widen the community and help with support for new architectures.
- Abstraction layers (though these have not been very successful so far)
- BLAS model as (ancient) successful example for coarse-grained implementations. For fine-grained, fast thread-spawning algorithms, there is a lack of good solutions other than re-implementing algorithms and re-coding.
- Domain-specific libraries?
- More funding for software and less focus on hardware?
- Workflows for heterogeneous hardware
- Investment in automated tools to translate algorithms at a high level of abstraction to the specific architecture of the hardware. Search space of implementations.
- In the long term, perhaps there will be a fundamental shift in the way we “train” computers, rather than programming them
- Integrated systems of hardware/software for dedicated applications

3.3.3 Leveraging Industrial Software Practices in Academia

Participants: Margo Seltzer, Gagan Agrawal, Matt Turk, Ed Maginn, Andrew Christlieb, Cyrus Unrigar, Benjamin Winjum, John Mellor-Crummey, Saday Sadayappan, Adam Updegrove, Person C, Dhabaleswar K (DK) Panda, James Howison, Ewa Deelman, Hakizumwami Birali Runesha, Reuben Budiardja, Robert J. Harrison, Inanc Senocak, Jason Leigh, Gary Olson, Ganesh Gopalakrishnan, Paul Butler, Gregg Musiker, Stephen Siegel, Sameer Shende, Ashish Gehani, Robert McLay, Ken Subratie

Issues/Motivation:

- Some industry software practices to draw upon:
    - Version control
    - Code review (e.g., Phabricator)
    - Stability of the code base
    - Unit testing
    - Regression testing

    - Change management
    - Documentation
    - Package management (professional distribution)
    - Issue tracking
    - Agile methods (daily scrum, sprints, pair programming, etc.)
    - Pair programming: less experienced programmers (mentees) are paired with more experienced mentors and they work on code together for several hours or for full days. After a period of this, the mentee works alone and mentors do code review.
- Software engineering process may not be specific to the types of software you are developing
    - Tools may be different
    - Practices carry over
    - This is true for high-performance computing and academia emulating industry
- "Employment relationship"
    - Spectrum of competencies for software development and academic mission
- Balancing between the different needs
    - For whom is it a core responsibility to review code and implement practices?
    - Is there sufficient buy-in from the investigator?
- What is in the best interest of the educator and the student?
    - Preparation for the workforce?
    - Focus on innovation?
    - "Were you really investing the right amount of time in code review?"
    - Is this the same across disciplines?
- How can we leverage the knowledge of big industries? How can we get closer to them? How do we learn from them?
    - Resources for learning industry practices:
        - Software carpentry (though this does not entirely cover software process issues)
        - XSEDE workshops?
        - CMU SEI workshops
        - Agile training: https://www.scrumalliance.org/certifications or http://www.collab.net/services/training/agile
        - Federal workforce trainings
        - Software engineering curricula, especially upper-level graduate software engineering capstones (example)
- Question about cost/benefit of these techniques for projects
- Concern that implementing software processes can be strong unfunded mandates
- Could include in student's coursework. Can software process skills be built into student methods training? Can departments can be convinced of this?
- What is fair to require of students? Can we view these as appropriate skills that give students an alternative career path?
- Is it more realistic to tie expectations of these "best practices" to "computational scientists" (not clear that the room acknowledges a difference between computer science and computational engineers)? Campus computing groups (research computing centers) can provide some of this work and projects can add a "line item" for

resources, (e.g., get 10 percent of a professional software engineer's time). Can software institutes play a role here?
- Can build in line-items for "performance engineering" by commercial consulting firms?
- What tools are there for code review?
    - https://en.wikipedia.org/wiki/List_of_tools_for_code_review
    - LLVM style formatter
    - https://en.wikipedia.org/wiki/Phabricator
- How do we plan for software sustainability and reuse in the face of Francis Timmes' last bullet "programming for the future hardware"?

Potential Solutions:
- Some of these, or all of these, are practices that are viewed in some research areas as core to the research practice and education of graduate students
- To obtain professional software support for development done by domain scientists, make sure small line items (for example, 10 percent of a full-time employee) are in the budget. It is up to universities to maintain a pool of professional software developers to support their faculty.

3.3.4 Code Correctness

Participants: Zach Byerly, Rene Gassmoeller, Geof Sawaya, Abani Patra, Andreas Stathopoulos, Anthony Danalis, David Schloen, Gerry Puckett, Ilya Zaslavsky

Issues/Motivation:
- Two key aspects of "good code:" correctness and quality
- Correctness depends on the context
- Do unit tests/tests for special cases ensure code correctness? On different systems?
- How do you evaluate if a code is correct? Stability of convergence?
- Is a bug that still leads to the correct result really a bug?
- Is it worth the speedup to decrease the code readability?
- Does everyone need to retest published software? Probably not.
- Reproducibility for large problems is difficult due to resource limitation
- How to deal with poorly written code?
- There is a long Wikipedia article about software quality, but what aspects matter for a project?
- We've had these discussions since the 1980s. What's new?

Potential Solutions
- Wrong results are unacceptable. But sometimes you do not know what is wrong.
- Try to find as many simple test cases as possible
- Compare across different codes
- It is healthy to have more than one community code to compare results
- If available, use observations to validate and verify
- Use established practices and workflows to validate codes and document the usage of code
- To convince developers to implement sufficient code verification, it is helpful to raise awareness of this need among users

- Raise awareness for efficiency in high-performance computing environments during the process of resource allocation (mostly for large scale proposals)
- Example: XSEDE monitoring services: monitor resource usage (I/O, FLOPS), start a dialogue with users

Next Steps:

- NSF should promote requirements for better software quality
- Adopt existing standards, or develop a new quality measure for high-performance computing codes possibly dependent on the community (correctness, efficiency, interoperability, learnability, etc.)
- Try to encourage and enforce better practices in co-workers and users, depending on the needs and standards in your area
- Are “computational architects” the solution? Not for doing all the work, but to plan the architecture.

3.3.5 Code Review

Participants: Steven Brandt, Simon Hettrick, Kyle Niemeyer, Amarda Shehu, John Clyne, Simon Hettrick, Kyle Chard, Yung-Hsiang Lu, Matthew Newville, Nancy Wilkins-Diehr, Steven Brandt, Alberto Shehu, Jonathan Wren, Paul Navratil, Rion Dooley, Chris Rapier

Issues/Motivation:

- Perfecting code requires striking a delicate balance between devoting resources to creating the code and devoting resources to review it. This can become challenging, especially for smaller projects where resources are tighter. There’s also the problem that some people are attracted to research development because they don’t want to work within a tightly constrained development environment or deal with code reviews, which can be intense. The counter analogy is that of people who complain about changing the oil in their car; to some extent, the review process can be painful, but it’s necessary. Small groups should attempt to take on new skills over time.
- There’s a degree of community education that needs to take place (for example, making the case for not just storing all the code in Dropbox). There can be a barrier to entry: if it takes a long time to learn, small groups may lack the resources to gain needed skills.
- Side discussion: “WTFs per minute” has been proposed as a quick measure of code quality and as a way to increase awareness of code quality issues, but this might not be the most appropriate approach and can be too confrontational, especially when you want to welcome new contributors. On the other hand, it could be a useful way of preparing people for the sometimes gladiatorial nature of code review.
- What is the purpose of conducting a code review?
    - Education: other members of team have to see code and consequently gain a greater understanding of the entire code base. There isn’t enough time to provide each developer with a tour of the code, so the code review gets around this problem
    - Finding problems: an extra pair of eyes on the code will typically help find some problems
    - Provenance: allows changes in the code to be associated with the rationale for why they were made

    - Tracking career development (potentially): one can see how a developer's coding has improved (or not) over a period
    - Improving commenting: a second person's confusion is a useful tool for finding the places where comments are needed
    - Catching other problems, such as developers being "creative" (and potentially offensive) with variable names, comments, etc.
- The process of code review:
    - In one example, a seven-member team used the following process: 1) Go to bug report, 2) make the change, 3) commit it to a location where it could be reviewed, 4) someone more senior reviewed it, 5) the original developer then explained how it works.
    - Automatic code review system (Gerrit: https://www.gerritcodereview.com) for GitHub. These systems are an absolute necessity for professional software development.
        - Drawbacks: large changes can be too much for people to take on board, so they generally wouldn't receive many comments. Sometimes the testing took so long that new problems had arisen by the time the fix was ready to be committed.
    - This process can be done using version control by providing the senior developers with the privilege to allow pull requests. However, managing pull requests can be prohibitive if you have a substantial project management infrastructure (e.g., testing infrastructure)
    - A similar if more rigorous review saw everything from variable names up to code structure being reviewed
    - Pair programming: well regarded, but resource heavy. Useful for identifying errors in code.
- The above processes are resource (people) heavy. Can we provide a system for people without access to these resources?
    - Another process is that fixes must pass continuous integration tests, are then automatically translated to GitHub pull requests, and are then sanctioned by a senior person. This system is used at Globus, which has roughly three services being developed simultaneously. Merge conflicts have not been a problem so far.
- Useful DevOps tools that are worth looking at:
    - Docker
    - Jenkins
    - Git blame
    - Git bisect
    - Tools for checking that coding standards are met
- The problem with using many new tools in one's DevOps/testing/reviewing process is that it substantially increases the learning curve of new people in your group. There's some balance here. PhD students might push back because it's not an important part of their work. New developers might welcome new tools as this broadens their skill base.

Potential Solutions:

- For small groups (and academic groups), attract people with beer and pizza. Keep it informal, and people will come to discuss code.
- Recommendations for code review (what needs to be in place):

    - A team (more than one developer, although there was some debate on this point)
    - Version control
    - Unit testing (some debate about whether or not this is fundamental)
    - Issue tracking
    - Some means of communicating with a community
    - Coding standard/code conventions

Next Steps:

- Write (or link to) a document about "What to do to make sure your code doesn't suck"
- Established groups should be encouraged to publish their coding review processes. Doesn't have to be a paper; blog posts are an easier way for people to access the information.
- We need a list of the software engineering skills—and places for people to acquire them—to allow better code reviewing practices to be put into place

Section 3.4 - Breakout Session #4: GIS and Soap Box

For the theme "Domain-specific Software Infrastructure," keynote speaker Shaowen Wang (University of Illinois Urbana-Champaign) delivered a presentation titled "Open CyberGIS Software for Geospatial Research and Education in the Big Data Era." Following the presentation, five participant-driven breakout sessions were held on topics related to this theme and other "soap box" ideas generated during the agenda scrum.

3.4.1 Code Trust: Why should others trust results of software they didn't write?

Participants: Gregg Musiker, Margo Seltzer, Andreas Stathopoulos, Shawn McKee, Adam Updegrove, Anthony Danalis, David Schloen, Amanda Shehu, Rene Gassmoeller, Glenn Martyna, Robert McLay

Issues/Motivation:

- How do you trust the results of software you didn't write (or what can you, as a software developer, do to help users trust your code)?

Challenges:

- How do you get users to believe that software is doing the right thing?
- Even if the user believes it, how do you get the public to believe in the results that are produced?
- How do you release a library that is not accurate (i.e., by design)?
- If you give users flexible software that they can modify and/or plug in, then how do you know whether what they are doing is valid?

Potential Solutions:

- Good software design
    - Tests that you ship:
        - Unit tests
        - Functional tests
        - Tests on known inputs

        - Installation/check test that identifies whether the software appears to be running or not
    - Understandable (modular)
    - User input testing that includes clear warnings (this idea got a lot of traction when we talked about policy makers using data to make decisions)
    - Document limitations
    - In the presence of a plug-in or extension architecture, provide a test structure for those plug-ins so that users can “validate” that their additions are “doing the right thing,” at least to some degree
    - But, always be aware that users use your software in a way that is quite different from what you intended
- Data provenance
    - The software should produce some kind of “description” of what the software did to produce the result
    - It would be terrific if the provenance were automatically processable to produce a readable, understandable description of what happened
    - Intermediate data and documentation of intermediate steps would be useful, but how do you provide such data in a way that the user can comprehend?
- Alternative methods to validate the results
    - Especially important in areas where the answers are, by definition, probabilistic
    - Demonstration on cases that have “known” correct results
    - Provide some kind of assurances:
        - Statistical properties
        - Images of a result
        - Error bounds
        - Some accompanying analytical model that at least provides bounds/ sanity checks on what the software is doing
- Reputation
    - This can be problematic (they might trust us too much)
    - They might be using it incorrectly, for example, using software for purposes for which the software is not intended/designed
    - Check your biases!
- What about trying to automatically detect misuse?
    - Collect data on all users that describe something about how the software is being used
    - Perform some kind of anomaly detection and flag outliers in order to warn users that they might be doing something questionable/problematic
    - This is potentially a really interesting topic for a proposal!
- Test your software on some local populations:
    - Undergraduates (they will always do the thing you least expect)
    - Senior researchers in the area (they will have high expectations)
- Redundancy
    - A central clearinghouse is fine, but having multiple implementations of different tools can provide the opportunity to validate across packages
- Improving the trust of users
    - Kinds of users
    - Technical versus administrative versus policy making

    - Expert user license!
    - Visualization that shows range of possibilities/error, with real numbers (visualizations without numbers can be very misleading)

3.4.2 Watershed Science Domain Software Institute

Participants: David Tarboton, Anthony Aufdenkampe, Stacy Lynn, Greg Tucker, Zach Byerly, Matt Anderson, Marlin Pierce

Issues/Motivation:

- Explore development of a Scientific Software Innovation Institute (S2I2) for water domain science
- Conceptualization award proposal?
    - Two were funded 2-3 years ago, but fizzled because of lack of community buy-in (See http://renci.org/research/water-science-software-institute/ and http://isees.nceas.ucsb.edu/)
    - NSF solicitations:
        - http://www.nsf.gov/pubs/2011/nsf11589/nsf11589.htm
        - http://www.nsf.gov/pubs/2010/nsf10050/nsf10050.jsp
        - http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503489
- We have all the connections to develop the right community buy-in
    - The Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI)
    - Critical Zone Observatories (CZO) network
    - Community Surface Dynamics Modeling System (CSDMS)
    - Natural Resource Ecology Laboratory
    - Citizen science around water
        - http://citsci.org/
        - http://wikiwatershed.org/
        - Model My Watershed
    - Ecology/Biology?
    - Social sciences?
- Need to broaden scope from “water domain” to “watershed science domain”?
    - We need the biggest tent possible without losing the focus on things related to fresh water
        - People live in watersheds
        - People farm soils in watersheds
        - The critical zone is organized best by watersheds
- Scope of problems to solve:
    - Flood modeling
    - Preserving drinking water
        - Modeling how land cover and land use and conservation/restoration affect water quality, similar to Model My Watershed
    - Restoration
    - Harnessing citizen science data
- Connecting to data

Potential Solutions:

- Need to rally community to support redirection of resources from core programs to software institute
- Need to get allies within NSF
    - GEO:EAR-surface processes

    - Social, Behavioral and Economic Sciences (SBE)
    - Engineering
    - EarthCube
    - GEO:ATM
- Need to get allies from outside NSF
    - Dan Katz suggested that first we decide what is needed, then approach all the agencies that are interested in such a scientific software institute
- Citizen Science
    - Put data in a place it can be used (examples from Birds and Astronomy)
    - Rajiv Ramnath suggested we contact Debra Estrin, UCLA TinyMote
- Rajiv Ramnath suggested:
    - Critical Resilient Interdependent Infrastructure Systems and Processes (CRISP) program
    - Innovations at the Nexus of Food, Energy and Water Systems (INFEWS) program (Bill Miller, science advisor)
    - SI^2
    - Research Coordination Network (RCN)
    - Perhaps first organize community (for example, by funding workshops)
    - Conceptualization awards are open this year. Submit anytime.
        - Need strong computing elements
        - Need strong domain elements

3.4.3 Toward a National Ecosystem

Participants: Matt Turk, Sameer Shende, Rion Dooley, Paul Navratil, Simon Hettrick, Benjamin Winjum

Issues/Motivation:

- If you leave NSF to do it you'll get something similar to that which they've done in the past (and in particular, likely something following the XSEDE model)
- National software ecosystem
    - Top down or bottom up?
    - Need to clarify what is meant by a software ecosystem; does it exist or not?
    - We dance around ecosystems like R, Python, "The other sage" (later comment: are we talking about languages, or tooling and common set of technologies?)
- Aspects of ecosystem: You have a sense of your software neighborhood, what is usable. This gives a greater sense of awareness of what else is being done.
- What do we already have?
- Seek commonalities:
    - Project support ecosystem, especially DevOps tools
- Questions: Do we see any of the SI^2 projects as upstream or downstream and responding to changes?
    - Eclipse PTP was SI^2-funded. "Ray Tracing library could output to Jason's Sage for display."
- Distributions? Open Virtualization Archive (OVA) files for high-performance computing Linux (appliance with tools, including commercial tools). This resolves a complex set of dependencies (and handles the updates).
- Could people also share their processes/experiences with software selection, or perhaps publish these?

- Can we build a useful software map?
    - Mike Hucka's could be useful as a software catalog
    - http://depsy.org/ (shows dependencies based on mining software mentions from papers, comes from those that built impactstory.org)
    - http://scisoft-net-map.isri.cmu.edu:7777/ (shows dependencies between software based on what tools were run together on the TACC supercomputers, using data from XALT, which gathers data from the modules system (ldmod).
    - Arfon Smith at GitHub
    - Docker images include inspect of dependencies
- Ecosystem "of what"? Tools, people, projects, "shared apis or shared functionalities"?
- Call for cataloguing and useful description of SI^2 tools, with some kind of phylogeny in terms of tools. How are things related to each other by distance?
- Suggestions for filling out a DOAP ("Description of a Project") which describes projects and the software stack that they have.
- Could NSF ("an interested party") help make connections?
- How many people who are not funded by NSF should be involved?
- DOE software catalogs: DARPA x-data, XSEDE repository, XSEDE software search, XSEDE.org/software
- Consensus that filling out useful descriptions of a project, including software locations, should be a minimum reporting requirement
- Proposal for next year's SI^2 PI meeting to bring along software developers/architectures and include a hackathon process. This would require a travel award process or possibly a "workshop grant." Sessions could then be organized by shared component use.
- Robert van de Geijn: We host a retreat that includes core developers, but also downstream users.

Training:

- Who has software engineers on their projects?
- Robert van de Geijn: People should have a sandbox area where a small "corner" of your software is well-described in teaching exercises. This has the potential to train and attract new contributors.
- How can we have commonalities in training?
- Examples of training approaches: CCamp ("how do we train the next generation of chemists?") How do we identify commonalities in the tools and thus needs for training?
- Can we drive assessments of tool usefulness using experiments in which domain scientists are asked to solve problems? Are time-limited experiments appropriate, or does that unnaturally drive people to take particular approaches (e.g., using high level front ends, rather than using building blocks)? People are generally not trained to think about the relationship between tooling and productivity. Interpretation: "everyone believes that they are an end-user until they are not;" one should not expect people to believe, at the start, that they need to compose from modules. What are the trade-offs here, how often do pre-packaged solutions actually deliver real results, or is it tweaks every time?
- How can you demonstrate the benefit of learning to work with the components, rather than just giving them tools? If we just give them pre-packaged solutions, then we don't train possible contributors.

Goals:

- Know what software decisions everyone is making; knowing who is in one's "nearby neighborhood" (upstream, downstream, user complements)
- Identify a phylogeny of our ecosystem, relationships, interdependencies and competition (usage, not funding)
- Draw users into becoming contributors (recognizing that this means having students interacting with the tools at a source code level)

Challenges:

- Are the PIs the best people to bring the relevant people together? Others in the projects often know more about the software architecture and sourcing decisions.
- Absence of funding in existing grants to bring people together
- Understanding motivation of students in a training context (including goals such as "time to science")

Next Steps:

- DOAP records, searchable database
- Explore and/or fund a workshop focused on hacking and collaboration (SSI-esque)
- Facilitate connections such as meetings or mailing lists between meetings

3.4.4 Cyber Fluids Institute

Participants: Inanc Senocak, Kyle Niemeyer, Boyce Griffith, Reuben Budiardja, Janos Sallai, Gerry Puckett

Issues/Motivation:

- Discuss the idea of creating a national Cyber Fluids Institute: a uniform, "plug-compatible" code infrastructure for modeling problems in computational fluid dynamics
- There is a bottleneck to implementing commonly used components/operations in computational fluid dynamics; we don't want or need to redesign the wheel!
- An institute could take charge of maintaining these components/building blocks. Individual groups could use these modules to build code for their specific applications.

Potential Solutions:

- Examples of models:
  - BLAS/LAPACK: www.netlib.org
  - Trilinos: https://trilinos.org
  - PETSc: http://www.mcs.anl.gov/petsc/
- A (SI^2) cyber fluids institute would collect and develop commonly used components and handle hardware-specific implementations (e.g., GPU), MPI, etc.
- Has there been a duplication of efforts within this community?
- "Basic" infrastructure (mesh/discretization technology)
  - deal.II finite element library: https://www.dealii.org
  - libMesh: http://libmesh.github.io
  - Chombo: https://commons.lbl.gov/display/chombo
  - SAMRAI: https://computation.llnl.gov/project/SAMRAI
  - p4est: http://www.p4est.org

- "Full-featured" infrastructure
    - Chombo/EBChombo
    - MOOSE Framework: http://www.mooseframework.com
        - Tightly coupled with finite element discretization technology
    - ASPECT: https://aspect.dealii.org/
        - Specifically designed for the geodynamics community
    - Uintah: http://uintah.utah.edu
        - Designed for chemical reactions, with some computational fluid dynamics applications. Development and capabilities were driven by specific interests of DOE funder/developers.
- Why didn't OpenFOAM come out of the United States? The computational fluid dynamics community widely looks to this (especially for student projects), but it is controlled by a (non-U.S.) private company. OpenFOAM is technically open source, but it follows a "service center" model (one-way code membrane, paid development/ consulting work). Fundamentally, software development is not driven by community desires/interests.
- Some solutions may exist—many or most developed at DOE labs—but the community does not find them easy to use. Therefore, they are not widely used within the wider fluid modeling community and thus perhaps not truly successful as open-source projects.
- Domain-specific language examples:
    - NCAR Command Language (for post-processing/visualization)
- Tools for building domain-specific languages:
    - XText (Java-like): https://eclipse.org/Xtext/
    - Groovy: http://docs.groovy-lang.org/docs/latest/html/documentation/core-domain-specific-languages.html
    - GME (graphical): http://www.webgme.org
- Perhaps a centralized center similar to the Computational Infrastructure for Geodynamics as a model for a similar center for the fluid dynamics engineering community would serve as a focal point for nurturing efforts to develop "plug-compatible" software for solving/modeling problems of interest to this community.

Next Steps:

- Develop a cyber fluids gateway, which includes components, such as finite difference, finite volume, finite element approaches, and others necessary for operations commonly used within this community
    - For example, reduce operations across mesh
    - Operations should be agnostic to the underlying mesh type (e.g., structured versus unstructured)
    - Should offer interoperable building blocks that can be used for different applications (e.g., Riemann solver, domain decomposition)
    - Should use high-layer abstraction (to avoid dealing with specifics of MPI, GPU, etc.)
    - An API should be developed that hides the complexity of the underlying libraries
- Invite the computer science community to help develop a domain-specific language as an eventual next step
- Should be based on what the fluid dynamics community wants (e.g., a wish list of components). Community run, possibly NSF funded?

3.4.5 RAPID Mechanism for Ebola and Other Crises

Participants: Xilun Chen, Teresa Head-Gordon, Xiyin Wang, Thomas Cheatham, Ashok Srinivasan, Shaowen Wang

Issues/Motivation:

- It helps if you already have a group in place to use Rapid Response Research (RAPID) grants. It is harder to start from scratch, because it is difficult to hire someone just for a year. People who want to put a team together may want to look at old SI^2 meeting attendees to identify the expertise available. One can also look at experimentalists who are seeking computational support. Some felt that they were not aware of the RAPID mechanism.
- Linking computation timing with funding was very useful. Estimating the time required was more difficult, though, because the information available in new crises is limited at the beginning.
- The RAPID mechanism has much potential for impact because it can generate positive press stories on application of the research to meet national needs. It provides Blue Waters visibility. We found some government organizations interested in the work. Their lawyers are sometimes concerned about any bad publicity they may get regarding dealing with crises.
- Data is often hard to get in the initial stage of a crisis. Software and models need to be designed with this constraint in place. Funding for maintaining a database of use to the community may be desirable.

Summary:

- Participants in this breakout session discussed the effectiveness of the RAPID mechanism in addressing national crises, the impact of work arising from such research, and the challenges faced in using this mechanism. Participants included two PIs who received RAPID funding from NSF's Computer and Information Science and Engineering (CISE), Advanced Cyberinfrastructure (ACI) to address the Ebola epidemic.
- PIs with RAPID funding concurred that this mechanism was effective in helping them address the potential Ebola epidemic. Integration of funding with allocation of time on Blue Waters enabled the computational work to start quickly, which is critical in dealing with national emergencies. Computation plays an increasingly important role in science-based solutions to problems facing society. At one end of the scale, computational simulations at the nano-scale can help produce fundamental understanding of viruses, leading to discoveries such as drugs to treat diseases. At the other end of the scale, software can help integrate models and data from a variety of domains to help policymakers identify suitable solutions to prevent or contain an epidemic at the national scale. RAPID funding is an effective catalyst in bringing together computer scientists with those from the application domain to address practical problems of national importance, because a crisis serves as a strong inducement to inter-disciplinary collaboration. ACI's focus on computation was particularly useful in helping repurpose existing research to create computational infrastructure and techniques that can be extended to new emerging epidemics.

- RAPID funding promises greater visibility than traditional funding because of its potential to generate positive press stories on application of research to national emergencies. PIs with funding received some publicity even in the early stages of the research, and their outreach activities to government entities generated much interest. It provides the public with an appreciation for investments in large computational resources, such as the Blue Waters machine.
- Given the opportunities with RAPID funding, there are also several challenges. An inherent difficulty with a new epidemic is the absence of sufficient knowledge and data. This makes it difficult to estimate the computational time required on resources such as the Blue Waters. Some flexibility in the time allocated can help with the research. In addition, software and models need to be designed taking into account the likelihood of inadequate data and uncertainties in it. It may also help if small amounts of funding are available to continue maintaining databases generated from federal funding, which could be of potential use to the nation in emergencies.
- Yet another difficulty is in creating an interdisciplinary group to solve the problem. Some participants felt that it helped if one already had a group in place; it is harder to start from scratch, because it is difficult to hire someone just for a year. People who want to put a new team in place could, perhaps, look at lists of previous SI^2 meeting attendees to identify the expertise available. One could also look at experimentalists who are seeking computational support; a national emergency could be an opportunity to initiate inter-disciplinary research to address real-world problems. It would also help if there were funding mechanisms that permitted the continuation of such research beyond the one-year term of RAPID.
- Some participants were not aware of the availability of the RAPID mechanism. As CISE participates in more RAPID programs, perhaps the computer science community will gain greater awareness of this mechanism and of the potential for computer science to contribute to dealing with national emergencies.

Section 3.5 - Breakout Session #5: Next Steps

This breakout session was devoted to expanding on four of the “Next Steps” discussions of some of the previous breakout sessions.

3.5.1 Large Group: Boot Camp and Resources

Participants: Yung-Hsiang Lu, Kyle Niemeyer, Shawn McKee, Janos Sallai, Benjamin Winjum, Margo Seltzer, Neil Heffernan, Ashish Gehani, Cyrus Umrigar, Gene Cooperman, Inanc Senocak, Hakizumwami Birali Runesha, Jason Leigh

Issues/Motivation:

- Software carpentry (too basic?): http://software-carpentry.org/
  - Version control, R, unit test, Python, MATLAB, make, AWS? XSEDE
  - Meta: video, should we contribute (do not re-create what is already available)
  - How to design software, architecture design, reusability, modularity
  - Parallel computing, parallelization (NVIDIA qwikLAB), need the scientific context

    - Case-based approach, Windows versus Linux, checkpointing, etc.
    - Machine learning (for scientists, not to design new machine learning solutions)
    - User interface design, visualization, web design, Trac (issue tracking, bug tracking)
    - Continuous integration
    - The sequence of actions (show a graph of progression)
    - Show some programs as context, but will not teach programming
- Is in-person boot camp a good idea? Are we ready?
- Online? In Person? EdX for Linux tutorials, MOOC, certificates, badges?
    - Ranking content, flipped classroom
    - Watch some online modules before coming to the boot camp
    - Where will the online materials be hosted?
    - Who will attend? Who will pay? How long?
    - Who will contribute the materials?

Potential Solutions:

- There is a list of recommended best practices for supported codes of the Computational Infrastructure of Geodynamics on different levels (minimum, recommended, target): https://geodynamics.org/cig/dev/best-practices/
    - http://swcarpentry.github.io/slideshows/best-practices/index.html#slide-0
- Discover online resources, rank them by quality, and compile a list
- Books about good approaches for software development
- Template for communication between domain scientists and computer specialists
- Create a tree (graphs) of modules needed by domain scientists

Next Steps:

- Create a shared document for people to suggest topics/modules
- Create a wiki site
- Use skiing difficulty colors (green circle, blue square, black diamond) for difficulty
- GitHub wiki

3.5.2 Large Group: Virtual Boot Camp

Issues/Motivation:

- How do you get the folks in your lab (or yourself) up to speed? This is mostly focused on getting domain scientists up to speed; there is an equally long list for helping the computer scientists get up to speed in a particular discipline. It would be great to produce a template that domain scientists could fill out to introduce their area to a computer scientist:
    - A list of terms
    - What math do you need us to know?
    - What do you mean when you say model?
    - What kind of computational proficiency does the standard person in your lab have?

Topics:

- From software carpentry:

    - Almost everything there
- Other stuff we should find:
    - Linux
    - How to get started/ structure a project: There is a list of recommended best practices for supported codes of the Computational Infrastructure of Geodynamics on different levels (minimum, recommended, target): https://geodynamics.org/cig/dev/best-practices/
    - How to build/design/architect software
    - How to use Amazon Web Services
    - How to use XSEDE
    - MPI/Open MPI
    - GPU programming
    - Hadoop
    - Data management
    - Transparent checkpointing
    - Data provenance
    - Need to provide a context for the work, some of which needs to be domain specific. Perhaps doing something like a domain-specific case study to walk through an existing project and talk about how various pieces fit together, how you select the right tool, etc.
    - Machine learning for scientists
    - Data visualization and user interface design
    - Tools: doxygen, Trac, Wiki, Jenkins
    - The University of Texas at Austin has a minor in computational science plus a bunch of training courses that range from half day to full week (intro to Python; intro to Linux) that are all available online
    - NVIDIA qwik labs
    - Check out Coursera, Udacity, EdX, Linux foundation online course

Next Steps:

- Everyone contributes a set of links of appropriate material
- GitHub organization created (SI^2 placeholder)

3.5.3 Keeping Software Alive

Participants: Hazel Asuncion, Matt Turk, Inanc Senocak, Sameer Shende, Adam Updegrove, Andreas Stathopoulos, Xiyin Wang, Bruce Berriman, Stacy Lynn, Matthew Newville, Nicholas Gottschlich, Gregg Musiker, Dmitry Pekurovsky, Paul Navratil, Kensworth Subratie, Kyle Chard, Zachary D. Byerly, Emilio Gallicchio, Anthony Danalis, Steven R. Brandt, Chris Rapier, Robert McLay

- Once there's no more money, how do we keep software alive? What happens when we no longer maintain it?
- Open sourcing is not a simple solution.
- Distinction: keeping software alive versus keeping it alive on life support

    - One approach: Freeze a given version of the software along with dependencies in a virtual appliance file, so that everything can be downloaded and used. "Known version that actually works."
- If you have an application for your software, you can keep it alive through funding rather than focusing on expansion, new development, etc. Keep it alive and active—adapting, but not "new." The software is not the product, the applications are.
- Can be part of the infrastructure in other grants (as "instrumentation")
- Example: A series of grants that keep things moving, once the funding for (patches to Linux kernel) development has finished. Industry supports by using, but not necessarily by funding, even at low levels.
- Is there tiering? Free, pro, etc.
- Evaluation of software: Do people and organizations utilize this? Should that factor into the evaluations of software and their impact/need?
- Another idea is to do crowdfunding
- "We're in this place, enabling science, but it's frustrating to have a bake sale." Next steps: How do we identify longer term funding sources for fundamental components? Or, long, low dollar amount maintenance grants.
- Maintaining software, especially security-based software, requires constant updating.
- Is this part of "faculty time"? "Academic time"? Some of the work that needs to be done is development and new functionality. These are things that grad students might find interesting and useful for their career, but maybe not.
- Data management plan can include software that is used, which could mean directly supporting software development
- One size doesn't fit all
- Robert Lass: "Frequently Forgotten Fundamental Facts about Software Engineering" suggests that maintenance can be 60 percent of software costs
- Some types of software require a considerable amount of porting
- Types of maintenance costs:
    - Adaptability to the environment
    - Bug fixing
    - New features
- Research Experience for Undergraduates grants are possible. Many can be hired, but they don't always work out. Using those types of resources can be good. Can be really hard for non-faculty.
- Can community building contribute to maintenance as well as new features?
    - Community building requires a lot of work
    - "Unpaid labor" can take advantage of folks
    - Testing can be very tricky, especially in open source and community projects
    - "Only happens when the other solutions are too painful, and so buy-in is powerful."
    - Best days are when somebody else answers questions
    - Putting it out to community
    - Building trust
    - Expectation that you'll be around
- What motivates? Pain of other solutions? What about redundancies and other solutions? How to capture and sustain partnerships with users/community in the long run?
- How can we write into the grant maintenance mode? Make things mission critical?

- Would it be useful to separate enhancements from pure maintenance? (for example, create version 2, but still support version 1)
- Need good documentation and tutorials, what the user might need to get started. Examples of how to modify the software.
- Containers that show the software well-installed and running?
- “People need to understand that open source software is free as in a free puppy, not as in a free beer.”
- “Reiss” profile of motivations. How does your community serve the motivations of others?
- Can splitting the code into different sections be worthwhile? Helps to have the code face different communities.
- Story about a package built in ANSI C and having a “quick start” guide to get started
- Projects that have undergone leadership transitions:
    - Enzo
    - Cactuscode.org (transition from Paul Walker to others)
    - HDF5 (the HDF group)
    - yt being taken over
- Does anyone want to maintain their software for the rest of their career? (Some expected to, as long as people are using it, hope to keep extending it, while others sought to hand it off)
- At least continuity of software is important, group ownership and passing it on

3.5.4 Next Workshop Suggestions

Participants: Amarda Shehu, John Clyne, Robert van de Geijn, Reuben Budiardja, Alberto Passalacqua, Mike Hucka, Matt Anderson

Ideas:

- Possible goals: Impart practical information to attendees that includes best practices, real-world examples of successful projects and addressing obstacles
- Talks by experts (e.g., Society of Women Engineers, DOE)
    - Bring the software engineering community into the meeting. One possible way of doing this is to bring a software engineer in to give a talk/demo (suggested person: Don Batori).
    - Note that PIs are not generally software engineers
    - DOE community has a stellar record of sustained software development. They have the labs, etc. What can we learn from that? (for example, Trilinos) Can we bring in Mike Heroux or Barry Smith to talk about best practices in scientific software engineering?
    - It may be desirable to examine how the DOE approaches sustaining software over the long term. What can be learned?
    - What does a project need to succeed?
        - Every single project needs a passionate believer in order to succeed.
- Talks by actual PIs that brought projects to completion
    - Hold a best practices session with successful PIs giving short testimonials and allowing for questions and answers
- Bring industry?

    - Is there a role for commercial vendors or industry representatives in this forum?
- Additional topics of interest for attendees not normally covered at this workshop
    - How many of the projects have partnered with industry?
        - Focus session on creating industrial affiliate programs
    - Discussion about leveraging intellectual property resulting from projects (e.g., Icorp, etc.)
    - Have a speaker about emerging hardware. What is out there, what is coming, what should we be aware of?
    - Have a speaker to give perspective on international software efforts. Suggestions include Chris Bishop (Darmstadt) and Neil Chue Hong (UK).
    - Session on how to incorporate pedagogical outreach in our community
    - Should we do a code review live in the workshop to illustrate the process?
- Less structure, more free format?
- Need a way to find out about what everyone else is doing. Suggestions include:
    - Reduce the number of breakout sessions to allow more free time for interaction
    - Hold a series of lightning presentations similar to elevator pitches, in which people give just one slide to show their domain, their software, etc.
    - Hold multiple poster sessions so presenters can stand in front of their posters
    - Use electronic posters instead of printed posters to allow movies and more interactive material

Section 3.6 - Conclusions

This section offers a summary of the conference outcomes from the perspective of workshop Co-PI Matthew Turk. Rather than a strictly objective summary, it is intended to offer an interpretation of the discussions and overall themes of the workshop.

The SI^2 PI meeting serves multiple purposes. Enabling cross-pollination of opportunities for technical collaboration around software is clearly a crucial goal, but it is by no means the only outcome. The organizing committee designed the workshop to facilitate discussion of enabling uptake of software, fostering community around software, understanding and applying best practices in software development, training and career paths for researchers, and a national ecosystem of software tools. The workshop sessions, and more importantly, the participating members of these sessions, provide dissemination of experiences as well as prescriptive suggestions for improving how scientific software projects are conducted. In this way, the workshop provided an enormously valuable opportunity for many of the PIs to learn from each other and to build social capital that can pay dividends long after the workshop has ended.

Most SI^2 grants are for three years, and the turnover in software projects necessitates that experiences are re-discussed; it's not a guarantee that lessons learned or conclusions reached one year can be strictly built upon, as the attendees the following year will be a slightly different selection of project leaders. While this can lead to a rehashing of material from previous years, it also leads to fresh ideas and new synthesis of opinions and ideas between groups. As an example, while past years had featured discussions of best practices for software development, this year's discussion of specifically testing for correctness was a new opportunity to address challenges specific to scientific software.

Perhaps one of the most useful aspects of the SI^2 PI meeting is that it brings together such disparate communities. The tendencies of the SI^2 program to attract projects that are heavily high-performance computing focused, as well as projects that target particularly deep domain problems through software, means that the synthesis of these two communities brings together different mindsets about scientific software (and both the value proposition and success metrics of that software), as well as different backgrounds in software engineering and design goals. This is an incredibly valuable opportunity for these two communities to cross-pollinate and understand how best to collaborate with each other.

The final breakout session included a discussion of what an ecosystem of SI^2 projects might look like. In a microcosm, this is what the workshop provides: the opportunity to develop collaborations, to guide needs and respond to changes, and to communicate about the technologies and communities that can be productive and helpful in enabling all of us to achieve our individual and collective goals.

SECTION 4.0 – PARTICIPANT FEEDBACK

Workshop participants were requested to provide feedback on the following three questions. Five participants provided written feedback.

1. *How did the Knowinnovation-facilitated, participant-driven style of this workshop work for you?*
    a. “It worked ok”

    b. “It worked great!”

    c. “This had a lot of positives. There was a lot of participation, and it allowed for discussion of some of the most pertinent topics for many participants. The only negative is that it didn't necessarily provide time to prepare for some of the discussions. It would've been nice to come in knowing that there were a couple of solidified sessions in which you could prepare for and provide more relevant information for.”

    d. “I think the conference ran smoothly, but I did not see huge differences from last year's.”

    e. “I attended the 2015 version of this event as well. I found that very rewarding, primarily due to the breakout format where I learned a lot from anecdotes presented by other participants.”

2. *Did you get out of this workshop what you wanted? Why or why not?*
    a. “… I feel comfortable telling you how illuminating the two SI^2 PI meetings have been for me. The "Open CyberGIS Software..." talk by Shaowen Wang was nothing short of eye opening.”

b. “I can give numerous examples of applications and approaches that I was simply unaware of prior to the meetings yet now inform my teaching, mentoring, and my participation in and support of new initiatives <here>, such as one in 'Big Data' currently being discussed.”

c. “Yes. Learned and found opportunities for collaboration!”

d. “I had hoped to learn a lot more about NSF's activities in the area, other participants' projects, and seed collaborations with others. Little of this occurred. I believe this may have been due to a combination of the agenda, the scheduling, and the facilitation process.

The issue was compounded by the complete absence of institutional memory regarding what was discussed last year (which would have been possible with organizer-defined questions - The Organizers apparently did not care).

Consequently, some of the breakouts spent significant effort re-hashing issues that were already discussed last year (since many participants had neither attended the previous workshop nor read the resulting report). Further, even if NSF decides to adopt suggestions from such workshops, the fact that it may take years to effect the changes means cognizance of previous suggestions and ongoing plans is important (and is only assured if organizers seed discussions with germane questions/foci).”

e. “Yes! I wanted to learn more about other projects going on, practices of other groups, and how we can improve ours. I learned a plethora of information in all of these categories.”

f. “Quite so. Exposed in practices and problems of other groups. Sometimes this gives new solutions, but more so it gives perspective. Networking was *excellent*.”

3. *Comments or suggestions for improvements for the 2017 SI^2 PI workshop organizers?*
   a. “It would be good if the poster session was split into two, so in each session 1/2 the PIs had the opportunity to move around and interact with the presenters. Also, suggest holding it during the day as rather than in the evening (more NSF people could attend then).”

   b. “Put posters up earlier”

   c. “As described above, I believe the organizers should not depend on the participants to define the agenda to the extent that this was done in 2016. A return to the balance struck between organizer-defined questions/foci and participant-driven breakout discussions may be more productive.

   Some attendees thought the best part of the workshop was the poster session, where they could learn about each others' work; this was left to the end of a 12-

hour day where people were tired. This session should be given a longer duration and a time slot that recognizes its importance.

The 12-hour and 4-hour breakup seems suboptimal. I understand the motivation for finishing early on the second day. However, several attendees thought a 12-hour first day was too long.

Since attendees come from all over the country, why does this workshop not rotate, through locations in different time zones/states? Expecting attendees from the West coast to arrive at the workshop at 4:00 a.m. Pacific Time is insensitive. Similarly, expecting West coast attendees to fly to the workshop on a holiday could easily be avoided."

d. "I know it is a quick meeting, and there isn't a lot of time, but it would be nice to have a very brief summary of what took place in the previous year, and what has been done to resolve issues brought up. I think this might help to make sure that things that have been brought up are followed through."

e. "Perhaps the poster sessions could start with each PI giving a 1-minute pitch on their work."

Participants were also polled to self-identify their roles. This table summarizes the 99 responses that were received.

Computer Science	42
Software Engineer	8
Chemistry	7
Physics	5
Geology	4
Interdisciplinary Scientist	3
Chemical Engineer	3
Materials Scientist	3
Bioinformaticist	2
Astronomy and Astrophysics	2
Mechanical Engineer	2
Sociology	2
Mathematician	2
Combustion	1
Archeology	1

Electrical Engineering	1
Better Software Advocate	1
Computational Linguist	1
Fluid Dynamics	1
Educational Psychologist	1
Data Scientist	1
Geography	1
Civil Engineering	1
Information Science	1
Programmer	1
Graduate Student	1
Network Research	1

ACKNOWLEDGEMENTS

This workshop was funded by the National Science Foundation through grant number 1606994. The Organizing Committee expresses its sincere appreciation to the National Science Foundation for their support. The committee extends a special thank you to all the attendees for participating in the workshop discussions and panel sessions, leading breakout sessions, taking notes, and contributing to this report. The interest in sharing dialogue among research communities and related projects is now stronger than ever as a result of this workshop. The outcomes provided will be of great value to the NSF and our respective research communities.

APPENDIX A: WORKSHOP ATTENDEES

126 participants registered and 98 attended.
*Denotes attendance in person. Some who registered were unable to attend due to inclement weather.

*Gagan Agrawal, The Ohio State University
Stan Ahalt, RENCI, University of North Carolina at Chapel Hill
*Matthew Anderson, Indiana University
*Hazeline Asuncion, University of Washington

*Anthony Aufdenkampe, Stroud Water Research Center
Dan Bates, Colorado State University
*Jerry Bernholc, North Carolina State University
*Bruce Berriman, California Institute of Technology
Raheem Beyah, Georgia Institute of Technology
Volker Blum, Duke University
James Bordner, University of California, San Diego
*Steve Brandt, Louisiana State University
*Richard Brower, Boston University
*Reuben Budiardja, University of Tennessee
*Paul Butler, National Institute for Standards and Technology
*Zach Byerly, Louisiana State University
*Kyle Chard, University of Chicago
*Thomas Cheatham, University of Utah
*Xilun Chen, Arizona State University
*Andrew Christlieb, Michigan State University
*John Clyne, National Center for Atmospheric Research
*Coray Colina, University of Florida
*Gene Cooperman, Northeastern University
*Peter Cummings, Vanderbilt University
Anthony Danalis, University of Tennessee
Damian Dechev, University of Central Florida
*Ewa Deelman, University of Southern California
*Rion Dooley, Texas Advanced Computing Center
*Timothy Dunne, Knowinnovation
*Peter Elmer, Princeton University
*Jose Fortes, University of Florida
*Emilio Gallicchio, CUNY Brooklyn College
*Rene Gassmoeller, Texas A&M University
*Ashish Gehani, SRI International
*Ganesh Gopalakrishnan, University of Utah
*Nicholas Gottschlich, University of Texas at Austin
Sol Greenspan, National Science Foundation
*Boyce Griffith, University of North Carolina at Chapel Hill
*Karl Gustafson, RENCI, University of North Carolina at Chapel Hill
*Ben Haley, Purdue University
*Robert J. Harrison, Institute for Advanced Computational Science, Stony Brook University
*Teresa Head-Gordon, University of California, Berkeley
*Neil Heffernan, Worcester Polytechnic Institute
*Simon Hettrick, Software Sustainability Institute
*James Howison, University of Texas at Austin
*Mike Hucka, California Institute of Technology
*David Hudak, Ohio Supercomputer Center
*Ray Idaszak, RENCI, University of North Carolina at Chapel Hill
David Irwin, University of Massachusetts, Amherst
*Shantenu Jha, Rutgers University
*Doug Johnson, Ohio Supercomputer Center

*Tom Jordan, Southern California Earthquake Center, University of Southern California
*Daniel S. Katz, National Science Foundation
Louise Kellogg, University of California, Davis
James Kirby, Naval Research Laboratory
*Robert “Mike” Kirby, University of Utah
Pablo Laguna, Georgia Institute of Technology
*Jason Leigh, University of Hawaii
Chris Lenhardt, RENCI, University of North Carolina at Chapel Hill
*Yung-Hsiang Lu, Purdue University
*Stacy Lynn, Colorado State University
*Edward Maginn, University of Notre Dame
Carlos Maltzahn, University of California, Santa Cruz
*Glenn Martyna, IBM Thomas J. Watson Research Center
Matthew Mayernik, National Center for Atmospheric Research
Tam Mayeshiba, University of Wisconsin, Madison
*Shawn McKee, University of Michigan
*Robert McLay, Texas Advanced Computing Center
*John Mellor-Crummey, Rice University
*Costa Michailidis, Knowinnovation
*Tom Miller, California Institute of Technology
Daniel Mosse, University of Pittsburgh
*Gregg Musiker, University of Minnesota
*Paul Navratil, Texas Advanced Computing Center
*Matthew Newville, University of Chicago
*Kyle Niemeyer, Oregon State University
*Gary Olson, University of California, Irvine
*Dhabaleswar Panda, The Ohio State University
*Alberto Passalacqua, Iowa State University
*Abani K. Patra, University at Buffalo
*Dmitry Pekurovsky, San Diego Supercomputer Center, University of California, San Diego
*Marlon Pierce, Indiana University
*Elbridge Gerry Puckett, University of California, Davis
James Pustejovsky, Brandeis University
*Rajiv Ramnath, National Science Foundation
*Chris Rapier, Pittsburgh Supercomputing Center
*H. Birali Runesha, University of Chicago
*P. (Saday) Sadayappan, The Ohio State University
*Janos Sallai, Vanderbilt University
*Geof Sawaya, University of Utah
*David Schloen, University of Chicago
*Margo Seltzer, Harvard University
*Inanc Senocak, Boise State University
*Amarda Shehu, George Mason University
*Sameer Shende, University of Oregon
Mark Shephard, Rensselaer Polytechnic Institute
*Steve Siegel, University of Delaware
*J. Ilja Siepmann, University of Minnesota

*Shava Smallen, San Diego Supercomputer Center
*Ashok Srinivasan, Florida State University
*Andreas Stathopoulos, College of William and Mary
*Ken Subratie, University of Florida
*David Tarboton, Utah State University
Douglas Thain, University of Notre Dame
George K. Thiruvathukal, Loyola University, Chicago
*Frank Timmes, Arizona State University
*Greg Tucker, University of Colorado, Boulder
*Matthew Turk, University of Illinois at Urbana-Champaign
*Selcuk Uluagac, Florida International University
*Cyrus Umrigar, Cornell University
*Adam Updegrove, University of California, Berkeley
*Robert Van de Geijn, University of Texas at Austin
Jan Verschelde, University of Illinois at Chicago
Jan Vitek, Northeastern University
*Rich Vuduc, Georgia Institute of Technology
*Ross Walker, University of California, San Diego
*Shaowen Wang, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
*Xiyin Wang, University of Georgia
Joannes Westerink, University of Notre Dame
*Nancy Wilkins-Diehr, San Diego Supercomputer Center
Laurie Williams, North Carolina State University
Philip A. Wilsey, University of Cincinnati
Benjamin Winjum, University of California at Los Angeles
*Jonathan Wren, Oklahoma Medical Research Foundation
*Ilya Zaslavsky, San Diego Supercomputer Center
Ye Zhao, Kent State University