OVERVIEW AND SUMMARY OF THE FIRST WORKSHOP ON SUSTAINABLE SOFTWARE FOR SCIENCE: PRACTICE AND EXPERIENCES (WSSSPE1)

AUTHORS TBD

Abstract. This document discusses the First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1). The content is based on the workshop itself, as recorded in a set of collaborative notes taken during the workshop, including discussion about the keynote presentations, three themed panels that focus on papers accepted by the workshop, other discussion, cross-cutting issues, use cases, and conclusions. It joins a nascent literature seeking to understand what drives software work in science and how the reward systems of science thereby shape the type of software work undertaken, including the extent to which developers are motivated to build software for the long-term, for the use of others, and whether to work collaboratively or separately. Unique perspectives were captured about issues such as documentation of APIs, software deployment, and development practices; software licenses; career paths for scientific software developers; identifiers for software through a mechanism such as a Digital Object Identifier or publication of a "software paper"; and evidence of software contribution and impact, recorded in online systems such as source code repositories like GitHub. For truly sustainable software there should be no endpoint, as the software products continue to be used and useful beyond the single institution, grant, and developer or development team.

papers and presentations and URLs should be references - see figshare-web in .bib file for URL style example

People who have volunteered to work on this report: (with * for actual contributors - if you contribute, add a * by your name)

The workshop organizers:

- * Daniel S. Katz ⟨dkatz@nsf.gov⟩
- Gabrielle Allen ⟨gdallen@illinois.edu⟩
- Neil Chue Hong ⟨N.ChueHong@software.ac.uk⟩
- Manish Parashar ⟨parashar@rutgers.edu⟩
- David Proctor ⟨dproctor@nsf.gov⟩

Additional volunteers:

- Chris Mattmann ⟨chris.a.mattmann@jpl.nasa.gov⟩
- * Ketan Maheshwari ⟨ketancmaheshwari@gmail.com⟩
- Marlon Pierce ⟨marpierc@iu.edu⟩
- Colin Venters ⟨C.Venters@hud.ac.uk⟩
- Suresh Marru ⟨smarru@iu.edu⟩
- Lynn Zentner ⟨lzentner@purdue.edu⟩
- Anne Elster ⟨anne.elster@gmail.com⟩
- * Shel Swenson ⟨mdswenso@usc.edu⟩
- Andy Ray Terrel ⟨andy.terrel@gmail.com⟩
- Abani Patra ⟨abani.patra@gmail.com⟩
- * Nancy Wilkins-Diehr ⟨wilkinsn@sdsc.edu⟩
- James Spencer ⟨j.spencer@imperial.ac.uk⟩
- * Frank Löffler ⟨knarf@cct.lsu.edu⟩
- * James Hetherington ⟨j.hetherington@ucl.ac.uk⟩
- * Sou-Cheng (Terrya) Choi ⟨sou.cheng.terrya.choi@gmail.com⟩
- * James Howison ⟨james@howison.name⟩

- * Bruce Berriman ⟨gbb@ipac.caltech.edu⟩
- Hilmar Lapp ⟨hlapp@nescent.org⟩
- * Marcus D. Hanwell ⟨marcus.hanwell@kitware.com⟩
- Lucas Nussbaum ⟨lucas.nussbaum@loria.fr⟩
- * Matthew Turk ⟨matthewturk@gmail.com⟩

The original document is https://docs.google.com/document/d/1eVfioGNlihXG_1Y8BgdCI6tXZKrybZgz5XuQHjT1oKU/edit?pli=1# (but can no longer be edited). Note that the original document has comments in addition to text.

Executive Summary

■ [TODO: 1 page summary goes here]

1. Introduction

The First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1, http://wssspe.researchcomputing.org.uk/WSSSPE1) was held on Sunday, 17 November 2013, in conjunction with the 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC13, http://sc13.supercomputing.org).

Because progress in scientific research is dependent on the quality and accessibility of software at all levels, it is now critical to address many challenges related to the development, deployment, and maintenance of reusable software. In addition, it is essential that scientists, researchers, and students are able to learn and adopt software-related skills and methodologies. Established researchers are already acquiring some of these skills, and in particular a specialized class of software developers is emerging in academic environments who are an integral and embedded part of successful research teams. The WSSSPE workshop was intended to provide a forum for discussion of the challenges, including both positions and experiences. The short papers and debates have been archived to provide a basis for continued discussion, and the workshop has fed into the collaborative writing of this document. An estimate of 90 to 150 participants were present throughout the day. Additional papers based on extended versions of workshop submissions are expected. The level of interest in the workshop has led the organizers, working with some of the submitters and attendees, to plan two additional workshops (at the 2014 SciPy and SC14 conferences) and to turn the workshop website into a community website that can be used as a focus for further discussion and progress. Additionally, a minisymposium that aims to further explore some of the key issues raised in WSSSPE is co-organized by a WSSSPE1 participant at the 2014 SIAM Annual Meeting on "Reliable Computational Science."

Before the WSSSPE1 workshop took place, the organizers self-published a paper [1] to document the process of organizing and advertising the workshop, collecting and reviewing the papers, and putting together the agenda. Section 2 is based on that publication. The remainder of this paper is based on the workshop itself, as documented in a set of collaborative notes taken during the workshop [2], including discussion about the keynote presentations (§3), the three panels (§4-6), cross-cutting issues (§7), use cases (§8), and conclusions (§9).

2. Workshop Process and Agenda

WSSSPE1 was organized in collaboration of the relatively small group of five organizers and a larger peer-review committee with 36 members. This committee had early influence on the organization, e.g., on the specific call for papers.

The workshop call for papers included:

> In practice, scientific software activities are part of an ecosystem where key roles are held by developers, users, and funders. All three groups supply resources to the ecosystem, as well as requirements that bound it. Roughly following the example of NSF's Vision and Strategy for Software [3], the ecosystem may be viewed as having challenges related to:
>
> - the development process that leads to new (versions of) software
> - how fundamental research in computer science or science/engineering domains is turned into reusable software
> - software created as a by-product of research
> - impact of computer science research on the development of scientific software and vice versa
> - the support and maintenance of existing software, including software engineering
> - governance, business, and sustainability models
> - the role of community software repositories, their operation and sustainability
> - the role of open source communities or industry
> - use of the software
> - growing communities

  - reproducibility, transparency needs that may be unique to science
- policy issues, such as
  - measuring usage and impact
  - software credit, attribution, incentive, and reward
  - career paths for developers and institutional roles
  - issues related to multiple organizations and multiple countries, such as intellectual property, licensing, etc.
  - mechanisms and venues for publishing software, and the role of publishers
- education and training

Based on the goal of encouraging a wide range of submissions from those involved in software practice, ranging from initial thoughts and partial studies to mature deployments, the organizers wanted to make submission as easy as possible. The call for papers stated:

> We invite short (4-page) position/experience reports that will be used to organize panel and discussion sessions. These papers will be archived by a third-party service, and provided DOIs. We encourage submitters to license their papers under a Creative Commons license that encourages sharing and remixing, as we will combine ideas (with attribution) into the outcomes of the workshop. An interactive site will be created to link these papers and the workshop discussion, with options for later comments and contributions. Contributions will be peer-reviewed for relevance and originality before the links are added to the workshop site; contributions will also be used to determine discussion topics and panelists. We will also plan one or more papers to be collaboratively developed by the contributors, based on the panels and discussions.

58 submissions were received, and almost all submitters used either arXiv [4] or figshare [5] to self-publish their papers.

A peer review process followed the submissions, where the 58 papers received 181 reviews, an average of 3.12 reviews per paper. Reviews were completed using a Google form, which allowed reviewers to choose papers they wanted to review, then to provide general comments and scores on relevance to the organizers and to the authors. The review reports enabled the organizers to decide which papers to associate with the workshop, and allowed the authors to improve their papers.

The organizers decided to list 54 of the papers as significant contributions to the workshop, a very high acceptance rate, but one that is reasonable, given the goal of broad participation and the fact that the reports were already self-published. The papers were grouped into three main categories, namely *Developing and Supporting Software*, *Policy*, and *Communities*. Each subject area was associated with a panel and discussion at the workshop.

The workshop itself consisted of two keynote presentations and the three panels/discussion sessions. The panels were organized based on a classification of the workshop submissions into three categories, following the themes of the call for papers as modified by the areas of the submissions. Three to four representatives from each submission category were appointed as panelists, and assigned to read a subset of the papers in that category and then discuss them in the panel.

3. Keynotes lead: Sou-Cheng (Terrya) Choi

The WSSSPE1 workshop began with two keynote presentations.

3.1. **A Recipe for Sustainable Software, Philip E. Bourne.**

The first keynote [6] was delivered by Philip E. Bourne of University of California, San Diego. Bourne is a basic biomedical scientist but has also formed four software companies. He co-founded PLOS Computational Biology [7] and helped develop the RCSB Protein Data Bank [8]. He is working on automating three-dimensional visualizations of cell contents and molecular structures, a problem that has not been solved and when done, would serve as a key function of software in biomedical sciences.

Bourne's keynote presentation was entitled "A Recipe for Sustainable Software," and developed based on his experiences. He emphasized that sustainability for software "does not just mean more

money from Government" (see also Section 7). Other factors to consider, he mentioned, include costs of production, ease of maintenance, community involvement, and distribution channels.

In places, Bourne said, development in science has improved thanks to open source and hosting services like GitHub [9], but for the most part remains arcane. He argued that we can learn much from the App Store model about interfaces, ratings, and so on. He also mentioned BioJava [10] and Open Science Data Cloud [11] as distribution channels. On a related note, Bourne observed a common evolutionary pathway for computational biology projects, from data archive to analytics platform to educational use, and suggested that use of scientific software for outreach might be the final step.

Bourne shared with the audience a few real challenges he encountered. First, also an anecdote, he has looked into reproducibility in computational biology, but has concluded that "I have proved I cannot reproduce research from my own lab" [12].

Another problem he experienced was staff retention with respect to private organizations which reward those combining research and software expertise (the "Google Bus"). However, he is a strong fan of software sustainability through public-private partnerships. He noted that making a successful business from scientific software alone is very rare: founders overvalue while customers undervalue. He noted that to last, an open source project needs a minimal funding requirement even with a vibrant community — goodwill only goes so far if one is being paid to do something else. He talked about grant schemes of relevance in the U.S., particularly with regard to technology transfer [13, 14].

He also had problems with selling research software: the technology transfer office in his university wanted huge intellectual property reach through, whereby they would get a share of profits from drugs developed by pharmaceutical companies who use the software. He was aware this was unrealistic but the technology transfer office insisted for a while. He wants to push a one-click approach for customers to purchase university-written software.

He then presented arguments on directly valuing software as a research output alongside papers, a common discussion within this field. An interesting reference provided by him is [15], which explores involving software engineers in the review process of scientific code.

On the notion of *digital enterprise*, where information technology (IT) underpins the whole of organizational activities, he contended that universities are way behind the curve. In particular, he highlighted the separation of research, teaching, and administration into silos without a common IT framework as a blocker to many useful organizational innovations: "University 2.0 is yet to happen." He spoke of a circumstance where someone had used an algorithm developed for computational biology in marketing. The role of an institution is important in this space. He argued that funders can help train institutions, not just individuals, in this regard.

He concluded with a reference to his paper [16] and argued that computational scientists "have a responsibility to convince their institutions, reviewers, and communities that software is scholarship, frequently more valuable than a research article," a point with which all authors strongly agree.

3.2. **Scientific Software and the Open Collaborative Web, Arfon Smith.** The second keynote [17] was delivered by Arfon Smith of GitHub.

Smith's talk started with an example from data reduction in Astronomy, where he needed to remove interfering effects from the device. He built a "bad pixel mask," and realized that while it was persistent, there was no way or practice of sharing this with the data. Consequently many researchers repeated the same calculations. He estimated that 13 person-years were wasted in this redundant calculation.

"Why didn't we do better?" Smith asked of this practice. He argued this was because we were taught to focus on immediate research outcomes and not on continuously improving and building on tools for research. He then asked, when we do know better, why we do not act any different. He argued that it was due to incentives and their lack: only the immediate products of research, not the software, are valued. He referenced Victoria Stodden's talk at OKCon [18] which he said argued these points well.

C. Titus Brown [19], a WSSSPE1 contributor and participant, argued that with regard to reusable software, "we should just start doing it." In this regard Smith replied that documentation should be "treated as a first class entity." He noted that the open source community has excellent cultures of code reuse, for example, RubyGems [20], PyPI [21], and CPAN [22], where there is effectively low-friction collaboration through the use of repositories. This has not happened in highly numerical, compiled language scientific software. An exception he cited as a good example of scientific projects using GitHub is the EMCEE Markov Chain Monte Carlo project [23] by Dan Foreman-Mackey and contributors.

He argued that GitHub's *Pull Request* code review mechanism facilitates such collaboration, by allowing one to code first, and seek review and merge back into the trunk later.

"Open source is . . . reproducible by necessity," Smith quoted Fernando Perez [24], explaining that reproducibility is a prerequisite for remote collaboration. He pointed out that GitHub could propel the next stage of web development, i.e., "the collaborative web," following on from the social web of Facebook.

In conclusion Smith reiterated the importance of establishing incentive models for open contributions and tool builders, for example, meaningful metrics and research grants such as [3]. He urged computational scientists to collaborate and share often their research reports, teaching materials, code, as well as data by attaching proper licenses.

4. Developing and Supporting Software lead: Marcus Hanwell, contributors: Suresh Marru

■ [TODO: section should include what was presented and what was discussed]

The panel on developing and supporting software examined the challenges around scientific software development and support, mainly focused on research groups that also produce code in various forms. There was widespread agreement that developing and maintaining software is hard, but best practices can help. Several participants stated that documentation is not just for users, and paying attention to API documentation, tutorials for building and deploying software, along with documented development practices can be very helpful in bringing new developers into a project efficiently.

There was discussion that backward compatibility is not always desirable, and it can be very costly. This must be balanced with the aims of a given project, and how many other projects depend on and use the code when backwards incompatible changes are to be made. There are many examples in the wider open source software world of strategies for dealing with this, and again best practices can go a long way to mitigating issues around backwards compatibility. Many projects live with sub-optimal code for a while, and may allow backwards compatibility to be broken at agreed-upon development points, such as a major release for a software library.

Communities are extremely important in software projects, and both their building and continued engagement need attention during the project life cycle. Several of the submitted papers discussed how communities have been built around projects, and what is needed to enable a project to grow. Among these are public source code hosting, mailing lists, documentation, wikis, bug trackers, software downloads, continuous integration, software quality dashboards, and of course, a general web presence to tie all of these things together. Questions were posed around users not answering the questions of other users. Several participants offered counterexamples, such as mailing lists where developers do not participate as much due to users being more active, or whether the "core team" can end up setting unrealistic expectations by doing too much. Team Geek■ [TODO: add cite] and Turk's paper on scaling code in the human dimension■ [TODO: add cite] discuss how development lists that are welcoming to people actually have many more people contributing.

Recruiting and/or retaining personnel in this area is hard, with one of the major reasons being no long-term career paths (especially when compared with industry). How should software development be balanced with research? It is apparent that things are beginning to improve, such as incentives for software development in the form of altmetrics, tenure committee consideration, and

NSF "products" vs "publications." It was noted it can be very difficult to measure where people are using small bits of your code.

There were 13 articles about different experiences in this area, but little about GUI testing, performance, scalability, or agile development practices. There were several unique perspectives about issues such as managing API changes, using the same best practices for software as data, and going beyond simply "slapping an OSI-approved license on code."

The question of what "sustainable" means in the context of software was raised (see §7.) The resources at http://oss-watch.ac.uk/resources/ssmm discuss what to look for when choosing software for procurement, or reuse in further software development. Regardless of the license and development model that will be used, the future of the project must be considered. Even if a particular piece of software satisfies today's requirements, will it continue to do so in the future? Important questions include whether the supplier will still be around in five years time, will it still care for its customers needs, will it be responsive to bug reports and feature requests? Should you tie your investment to a single supplier using a proprietary product, or ensure the project uses an OSI-approved license, and can outlive any single entity if the software is still important.

What is the overarching goal of sustainability? Is it reproducible science, persistence, quality, something else? How should success be measured in this context? Is there some metric that can be used to determine when you have achieved sustainable software, or is this an ongoing process with no clear endpoint. For truly sustainable software there should be no endpoint, as the software products continue to be used and useful beyond the single institution, grant, and developer/development team. Sustainability must be addressed throughout the project life cycle.

What about actual software engineering principles, such as modularity and extensibility? This is how industry maintains software, and ensures it continues to be useful. Often, rewriting software is considered to be too costly, but with a modular design it can be kept up to date. Extensibility is expected to keep it relevant, if built into the project. One counterpoint raised by Jason Riedy was that trying to take advantage of the latest and greatest hardware often makes this painful, hence the lack of papers mentioning "GPUs and exotic hardware."

The question of whether funders, such as the NSF, can mandate software plans in much the same way as they do data management plans, was raised. Daniel Katz responded that software is supposed to be described as part of the NSF data management plan, and that in NSF's definition, data includes software. A comment from Twitter (@biomickwatson) raised the issue that this requires reviewers and funders who understand the answers that are given in these plans. Daniel Katz responded that in programs focused on software or data this can be done effectively, but agreed that in more general programs this is indeed a problem.

The papers submitted to this panel, and several others, include lots of general recommendations for processes, practices, tools, etc. One of the papers suggested that a "Software Sustainability Institute" should be vested with the authority to develop standardized quality processes, a central common repository, central resources for services and consulting, a think tank of sorts, and a software orphanage center. The idea of one common repository received some resistance, with so many compelling alternatives available, e.g. Bitbucket or GitHub. The point for centralization of communication/point of contact was seen as reasonable, with the statement that "vested with authority" is perhaps too strong, but "providing tools if needed" might be more appropriate.

Some of the questions raised after the panel discussion are:

Rather than teaching developers about domain science, or domain scientists about software development practices, why don't we teach both communities to collaborate more effectively? Can this be done without teaching each side a little of the other to enable communication, with a response that it really is not two binary communities, but a spectrum with useful roles in the middle. The question was raised if a developer can be effective without being part of the domain community, with responses that this really depends on the specific problem—translators and T-shaped people are important. Why aren't academic communities taught how to evaluate cross-disciplinary work well?

What is the role for the growing field of team science? There is overlap between the communities, with support for virtual organizations, tool development, etc. How can we make time in an already crowded schedule to introduce these topics to students? Should they be introduced through lab classes as in "real sciences"?

Are there significant differences in projects that have been running for 1, 3, 5, or 10+ years? Are there shared experiences for projects of a similar stage of maturity? It was noted that computing and communication have changed significantly over the past decade, and many of the experiences are tied to the history of computing and communication. See the history of GCC, Emacs, or the Visualization Toolkit for examples. Others felt that computing has changed less, but communication and the widespread availability of tools has. It was noted that email lists, websites, chat rooms, version control, virtual and physical meetings are all over 20 years old.

There was debate that while some of the basics of computing may be fairly similar tools commonly used for computing have changed quite significantly. Reference was made to Perl, which was commonly used, giving way to whole new languages, such as Python, for gluing things together and how this induces many students into entirely rewriting the scaffolding, leaving the old to rot and the experiments to become non-reproducible as the tools change. Jason Riedy stated that he had been guilty of this in the past. There was discussion of this tendency along with the enormous differences in the speed and ease of sharing—having to ship tapes around (GCC, Emacs, etc) as opposed to the immediate sharing of the latest development online, using revision control systems like CVS, Subversion, Git, Mercurial, Bazaar, etc.

The question was also posed as to whether the distinction between researcher and developer is sensible, with James Hetherington commenting that in the UK a more nuanced view of research software engineers and researcher developers is examined. Should this be less of a contract relationship, and more of a collaborative relationship? This is also at the core of the business model that Kitware presented in its submission to the workshop. Are other ingredients missing such as applied mathematicians? Should this be defined more in terms of skill sets rather than roles and/or identities? This builds on the comments from Vaidy Sunderam that scientists are generally good writers, and have mathematical skills, so why can't they learn software engineering principles?

Miller commented that all of the infrastructure that sits around a new algorithm that we need to make it useful and sustainable requires different skill sets than the algorithm developer. Friere commented that there are no good career paths for people with broad skills, no incentives for them to continue in these roles. There was debate around people doing what interests them, and learning computing leaves people cold, but is it that it leaves the people who find career paths in academia cold versus the full spectrum of people involved in research? Is this also caused by poor teaching, or because the benefits for doing this are perceived as too small? It could also be attributed to their focus being on science, not software engineering, or do people with the passion for software engineering in science simply have no viable career path and either adapt or seek out alternate career paths?

Papers.

Development Experiences.

- Mark C. Miller, Lori Diachin, Satish Balay, Lois Curfman McInnes, Barry Smith. Package Management Practices Essential for Interoperability: Lessons Learned and Strategies Developed for FASTMath [25]
- Karl W. Broman, Thirteen years of R/qtl: Just barely sustainable [26]
- Charles R. Ferenbaugh, Experiments in Sustainable Software Practices for Future Architectures [27]
- Eric G Stephan, Todd O Elsethagen, Kerstin Kleese van Dam, Laura Riihimaki. What Comes First, the OWL or the Bean? [28]
- Derek R. Gaston, John Peterson, Cody J. Permann, David Andrs, Andrew E. Slaughter, Jason M. Miller, Continuous Integration for Concurrent Computational Framework and Application Development [29]

- Anshu Dubey, B. Van Straalen. Experiences from Software Engineering of Large Scale AMR Multiphysics Code Frameworks [30]
- Markus Blatt. DUNE as an Example of Sustainable Open Source Scientific Software Development [31]
- David Koop, Juliana Freiere, Cláudio T. Silva, Enabling Reproducible Science with VisTrails [32]
- Sean Ahern, Eric Brugger, Brad Whitlock, Jeremy S. Meredith, Kathleen Biagas, Mark C. Miller, Hank Childs, VisIt: Experiences with Sustainable Software [33]
- Sou-Cheng (Terrya) Choi. MINRES-QLP Pack and Reliable Reproducible Research via Staunch Scientific Software [34]
- Michael Crusoe, C. Titus Brown. Walking the talk: adopting and adapting sustainable scientific software development processes in a small biology lab [35]
- Dhabaleswar K. Panda, Karen Tomko, Karl Schulz, Amitava Majumdar. The MVAPICH Project: Evolution and Sustainability of an Open Source Production Quality MPI Library for HPC [36]
- Eric M. Heien, Todd L. Miller, Becky Gietzel, Louise H. Kellogg. Experiences with Automated Build and Test for Geodynamics Simulation Codes [37]

Deployment, Support, and Maintenance of Existing Software.

- Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, Frédéric Suter. SimGrid: a Sustained Effort for the Versatile Simulation of Large Scale Distributed Systems [38]
- Erik Trainer, Chalalai Chaihirunkarn, James Herbsleb. The Big Effects of Short-term Efforts: A Catalyst for Community Engagement in Scientific Software [39]
- Jeremy Cohen, Chris Cantwell, Neil Chue Hong, David Moxey, Malcolm Illingworth, Andrew Turner, John Darlington, Spencer Sherwin. Simplifying the Development, Use and Sustainability of HPC Software [40]
- Jaroslaw Slawinski, Vaidy Sunderam. Towards Semi-Automatic Deployment of Scientific and Engineering Applications [41]

Best Practices, Challenges, and Recommendations.

- Andreas Prlić, James B. Procter. Ten Simple Rules for the Open Development of Scientific Software [42]
- Anshu Dubey, S. Brandt, R. Brower, M. Giles, P. Hovland, D. Q. Lamb, F. Löffler, B. Norris, B. O'Shea, C. Rebbi, M. Snir, R. Thakur, Software Abstractions and Methodologies for HPC Simulation Codes on Future Architectures [43]
- Jeffrey Carver, George K. Thiruvathukal. Software Engineering Need Not Be Difficult [44]
- Craig A. Stewart, Julie Wernert, Eric A. Wernert, William K. Barnett, Von Welch. Initial Findings from a Study of Best Practices and Models for Cyberinfrastructure Software Sustainability [45]
- Jed Brown, Matthew Knepley, Barry Smith. Run-time extensibility: anything less is unsustainable [46]
- Shel Swenson, Yogesh Simmhan, Viktor Prasanna, Manish Parashar, Jason Riedy, David Bader, Richard Vuduc. Sustainable Software Development for Next-Gen Sequencing (NGS) Bioinformatics on Emerging Platforms [47]

4.1. **Research or Reuse?** Discussion of software produced as a by-product versus software developed for reuse. How does this change the project, who develops the code, growth beyond 1–2 person projects to larger projects with diverse set of consumers and contributors. Modular and extensible code versus working for current research problem—tackled by same people, or a collaborative team?

4.2. **Defining Sustainability for Scientific Software.** What is sustainable? What are the best practices, can workshops fund meetings to help define and improve best practices in these areas?

Software plans from the funding agencies? This topic is expanded upon in §7, since it was discussed up in multiple parts of the WSSSPE1 workshop.

4.3. **Training Scientists to Develop and Support Software.** Discussion of the evolving role of scientists, and/or others that fill these roles.

4.4. **Software Process, Code Review, Automation, Reproducibility.** There are a lot of tools out there, but few are currently used. Look at what some projects have found successful, and how to automate as much as possible to reduce additional overhead.

4.5. **Software and Data Licensing.** Its impact on how and where research products are used, who can collaborate and what patterns have worked/not worked in existing communities.

4.6. **Funding, Sustainability Beyond the First Grant/Institution.** How initial software development is funded, moving to follow up projects, maintenance, community growth, collaborating with other institutions, labs, industry, internationally. Contract relationship versus collaborative development between scientists and software developers.

4.7. **Training Others to use Software.** Who creates training materials, how are they distributed, integration into courses when projects see very wide application in research.

5. Policy lead: Colin Venters, contributors: James Howison, Hilmar Lapp

Papers.

Modeling Sustainability.

- Coral Calero, M. Angeles Moraga, Manuel F. Bertoa. Towards a Software Product Sustainability Model [48]

  They examine existing ISO standards about software quality. These standards include attributes of the software but currently do not discuss sustainability, in terms of both meanings of sustainability (energy efficiency and what they call perduability, the software "lasting over time". In terms of lasting over time, they highlight characteristics of reliabiity, maintainability (e.g., modularity, the degree to which the code is understandable, testable and modifiable) and the degree to which the code can be adapted.
- Colin C. Venters, Lydia Lau, Michael K. Griffiths, Violeta Holmes, Rupert R. Ward, Jie Xu. The Blind Men and the Elephant: Towards a Software Sustainability Architectural Evaluation Framework [49]
- Marlon Pierce, Suresh Marru, Chris Mattmann. Sustainable Cyberinfrastructure Software Through Open Governance [50]

  They argue that the way that a project is run relates to whether it can continue to serve its users. In particular they argue that because sustainability relates to having ongoing resources, governance must be open to receive diverse input and thus have the potential to "transform passive users into active contributors".

  They also argue that open source principles have been mis-applied by the research software community.

  Their framework puts together three parts supported by open software: operations and support, escience research and scientific collaboration.

  They define open governance as "project deliberations on open, archivable forums ... with votes carried out asynchronously over a period of time ..." This, presumably, contrasts with how governance is done in research software projects at present, through offline meetings amongst grant funded groups, for the very large part.
- Daniel S. Katz, David Proctor. A Framework for Discussing e-Research Infrastructure Sustainability [51]

Their framework looks at (for each software element): technical (how does it work with other components), social (who is building it, who is using it) and political context (who is funding it).

The propose three axes on which to locate particular pieces of software: temporal extent over which software should persist, spatial extent (particular lab to global context), Purpose (particular use to general use.). They also discuss resourcing models: 1) commercial for profit, 2) open source, 3) closed partnerships and 4) funded projects. An additional model can function during a sustaining phase: open source with paid support. Finally they introduce two models of governance, top-down and bottom-up.

They hypothesize that each of these models will be appropriate for a different location on their axes and look to additional research to make the connections (for example, government funding might be most appropriate for longer time scales).

- Christopher Lenhardt, Stanley Ahalt, Brian Blanton, Laura Christopherson, Ray Idaszak. Data Management Lifecycle and Software Lifecycle Management in the Context of Conducting Science [52]
- Nicholas Weber, Andrea Thomer, Michael Twidale. Niche Modeling: Ecological Metaphors for Sustainable Software in Science [53]

Credit, Citation, Impact. The policy panel and discussion addressed the question of recognition of work on scientific software and linked that recognition to questions of reward and thus motivation for particular kinds of work on scientific software. In short, software work in science was seen to be inadequately visible in ways that "count" within the reputation system underlying science. In his paper for this workshop, Katz placed software work along with other "activities that facilitate science but are not currently rewarded or recognized" [51]. Priem and Piwowar argued for the need to "support all researchers in presenting meaningful impact evidence in tenure, promotion, and funding applications." [54]. Kneply et al. argued that the visibility of software that supported a piece of science "can have detrimental effects on funding, future library development, and even scientific careers." [55].

These papers, and the discussion at the workshop, join a nascent literature seeking to understand what drives software work in science and how the reward systems of science thereby shape the type of software work undertaken, including the extent to which developers are motivated to build software for the long-term, for the use of others and whether to work collaboratively or separately [56, 57, 58]. Software work is not only motivated by direct citations, but the visibility of software work in the literature is important to those who write software used in science.

Papers and discussion concentrated in three areas: How ought software work to be visible, what are the barriers to its visibility, and what can be done to make it more visible?

Most of the papers in this area focused on visibility of software in scientific papers, since scientific papers are the most widely accepted documentation of achievement in science. It was noted that there appear to be no widely accepted standards on how the use of software towards a paper ought to be mentioned, that journals, citation style guides and other guides to scientific conduct are vague about how to describe software. To address this, papers spoke of a need for a fixed identifier for software, either directly through a mechanism such as a Digital Object Identifier [51, 55] or via a published paper written to document the software (and perhaps its creation), a "software paper" [59].

Another approach is to try to reduce the difficulty of citing software for authors, acknowledging that authors are often working with software that itself wraps other software and therefore hides that software. Knepley et. al. approach this by proposing a mechanism by which the software itself, after it has run, provides the user with a set of citations that are relevant to the code actually run. They describe a prototype implementation whereby the citations are embedded in libraries and reported along with the results, via a commandline interface [55]. Discussion highlighted the difficulty that attempting to acknowledge the contributions of all pieces of dependent code within a paper faces the difficulty of creating very long citation lists, straining the analogy of code used to papers

cited. Katz approaches this issue by proposing a system of "transitive credit," recording dependencies and relative contributions outside particular papers, relieving authors from the responsibility of acknowledging each and every dependency. Instead authors would acknowledge the percentage contribution of the software they used directly and an external system would then be able to recursively allocate that credit to those who had provided dependencies. Finally Priem and Piwowar argued that machine learning techniques could examine the body of published literature and extract mentions of software, coping with the multitude of informal ways in which authors mention software they have used [54]. Discussion included turning the emphasis from asking users to improve their citation of software contributions, to ask how projects producing software might monitor the literature to improve their ability to show impact. Michael McLennan described the approach taken by the NanoHub project to scan the literature using keywords and names of known users to discover papers that are likely to have used their software and platform, then to have graduate students read each paper, highlighting any mention and perhaps following up with the authors to locate stories of impact. Work since the workshop has described this process at *publications.wikia.com*.

Acknowledgement in science does not come only in publications, of course. A key location for visibility might be in the grant funding process, both in bio-sketches and in grant project reporting. Some progress has been made here. Representatives of the NSF at the meeting emphasized these opportunities and encouraged participants to take advantage of them. Nonetheless, there was skepticism that peer review panels would value these contributions in the same ways as publications).

Priem and Piwowar argued for additionally looking beyond publications and drawing on evidence of contribution, and impact, recorded in other online systems, such as source code repositories like github, both code contributions, downloads of code and dependencies as well as conversations about software (mailing lists, twitter and beyond) [54]. They argue for providing scholars with flexible resources so that they can tell their own stories in the manner most appropriate for them and their audiences, a principle of the "alt.metrics" approach.

■ [TODO: convert this sketch to discussion of policy options] Opportunities for Policy interventions: Funding agencies Journals. Success of data policies, growth of software policies. Promotion and Tenure. Follow lead of Science of Team Science in surveying policies at universities and other institutes.

■ [TODO: review PiratePad notes for other aspects of credit-giving discussed.]

- Matthew Knepley, Jed Brown, Lois Curfman McInnes, Barry Smith. Accurately Citing Software and Algorithms Used in Publications [55]
- Jason Priem, Heather Piwowar. Toward a comprehensive impact report for every software project [54]
- Daniel S. Katz. Citation and Attribution of Digital Products: Social and Technological Concerns [60]
- Neil Chue Hong, Brian Hole, Samuel Moore. Software Papers: improving the reusability and sustainability of scientific software [59]

In addition, the following paper from another area were also discussed in this area.

- Frank Löffler, Steven R. Brandt, Gabrielle Allen and Erik Schnetter. Cactus: Issues for Sustainable Simulation Software [61]

Reproducibility.

- Victoria Stodden, Sheila Miguez. Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research [62]

Implementing Policy.

- Randy Heiland, Betsy Thomas, Von Welch, Craig Jackson. Toward a Research Software Security Maturity Model [63]
- Brian Blanton, Chris Lenhardt, A User Perspective on Sustainable Scientific Software [64]
- Daisie Huang, Hilmar Lapp. Software Engineering as Instrumentation for the Long Tail of Scientific Software [65]

- Rich Wolski, Chandra Krintz, Hiranya Jayathilaka, Stratos Dimopoulos, Alexander Pucher. Developing Systems for API Governance [66]

5.1. **Career Tracks for Scientific Software Developers.** How to ensure software is sustainable by ensuring there are career paths for developers.

What are the possible career paths for a specialist software developer working as part of a scientific research group?

6. Communities lead: Matthew Turk, contributors: Nancy Wilkins-Diehr, Chris Mattmann, Suresh Marru, Frank Löffler, Andy Terrel

■ [TODO: section should include what was presented and what was discussed]

There were a number of papers categorized under the Communities banner and so two presenters each summarized half of the submissions in this area.

6.1. **Communities Part 1.** This collection of papers was summarized by Karen Cranston.

Drawing on experiences from high-energy physics, [67] proposed developing teams of technical specialists able to overcome a lack of coordination between projects. Maximization of scientific output requires maximizing the usability of the scientific codes while minimizing the cost of developing and supporting those codes. This included targeting different architectures for their software to be deployed, as well as coordination between technically-focused individuals and usage of a common scripting language between projects. Instead of fragmenting the development of simulation codes across institutions, the paper suggests that a cohesive strategy reducing duplication and increasing coordination will broadly increase the efficiency across institutions. The approach proposed is of de-fragmenting the existing ecosystem in a non-disruptive way.

The paper by Maheshwari et al. [68] focuses on “technology catalysts" and their role in the modern scientific research process. A technology catalyst can be defined as a role played by an individual with a know-how of technological advancements, tasked with user-engagement with a goal of enacting scientific or engineering applications, and using suitable tools and techniques to take advantage of technological capabilities thus benefiting applications.

One of the tasks of technology catalysts is to seek community collaborations for new applications and user engagement thus benefiting both: science, by effective running of scientific codes on computational infrastructure, and technology, by conducting research and seeking findings for technology improvement. The particular engagements described in the paper came up from authors work as postdoctoral researcher at Cornell and Argonne. Interaction with the scientific communities in both institutions resulted in these collaborations.

In [69], the authors reflect on the 15-year history of open source software development at Kitware. In particular, they focus on their success at growing their community of users through enabling multiple channels of communication, directly reaching out to individuals, and lowering the barrier to entry for contributions. This involves providing clear, test-oriented and review-based mechanisms for evaluating contributions, permissive licenses, and a service-based model for sustaining development. This model enables Kitware to review both public funding, as well as private funding to support improvements and targeted developments of the software.

At NESCent, a combination of in-house informatics individuals and domain scientists collaborate to develop software to study evolutionary science. The report [70] studied the success of a “hackathon” model for development, where short-form, hands-on events combining users, researcher-developers and software engineers targeted specific code improvements. From this experiment, the authors identified several key outcomes as well as lessons-learned – specifically, the co-localization of developers was seen as having a strong impact, enabling casual conversation that led to discrete outcomes. The formation of the discussion mailing list, specifically in response to the social capital built at the hackathon, was seen as spurring on longer-term benefits to the community and fostering sustainability.

[71] addresses the success of the rOpenSci project in developing collaborations supporting tool development for Open Science. This software collective, organized around the statistical programming environment R, develops access mechanisms for data repositories and attempts to reduce the barrier to entry for individuals wanting to access data repositories and study the data contained therein. The collective fosters direct collaboration between individuals and data providers, designed to "train academics in reproducible science workflows focused around R." The two central challenges to this goal were identified as engagement of existing users within ecology and evolutionary biology (EEB), and how the community could make inroads and traction in other disciplines. Currently, the collective is exploring addressing these challenges through use of social media and holding workshops and hackathons. This helps to both raise the profile of the collective within EEB and in other domains. However, the overarching challenge identified in the paper was that of incentivizing maintenance of software, which is difficult in academia.

6.2. **Communities Part 2.** Nancy Wilkins-Diehr summarized 6 papers in this part.

Christopherson et al. [72] outlines the degree to which research relies on high quality software. There are often barriers and a lack of suitable incentives for researchers to embrace software engineering principles. The Water Science Software Institute is working to lower some of these barriers through an Open Community Engagement Process. This is a 4-step iterative development process that incorporates Agile development principles.

- Step 1: Design - thorough discussion of research questions
- Step 2: Develop working code
- Step 3: Refine based on new requirements
- Step 4: Publish open source

Christopherson reports on the application of Steps 1-3 to a computational modeling framework developed in the 1990s. Step 1 was realized as a 2-day, in person specifications meeting and code walkthrough. Step 2 was a 5-day hackathon to develop working code, and Step 3 was a 3-day hackathon to refine the code based on new requirements. The team worked on small, low-risk units of code. It was challenging, revealed unanticipated obstacles, programmers had to work together, and experimentation was encouraged.

The paper summarized recommendations to others wishing to engage in this or a similar process, starting small and gradually building toward more complex objectives. This is consistent with Agile development. Refactor before adding new functionality. Approach development as a learning experience. Welcome experimentation, and treat mistakes as a natural part of the learning process. Repeat Step 1 activities before all hackathons to develop consensus before coding. This allows coding to be the focus of subsequent hackathons. In higher risk situations, provide additional time for Step 1 activities. The Christopherson team recommends a minimum of two months. Ensure any newcomers receive some form of orientation prior to the hackathon, such as a code walkthrough or system documentation. Co-locate rather than collaborating remotely whenever feasible.

Pierce et al. [73] described how science gateways can provide a user-friendly entry to complex cyberinfrastructure. The paper opens with a description of the explosion of use of just a few gateways. Over 3.5 years, more than 7,000 biologists have run phylogenetic codes on supercomputers using the CIPRES Science Gateway. In 1 year, over 120 scientists from 50 institutions used the UltraScan Science Gateway, increasing the sophistication of analytical ultracentrifugation experiments worldwide. The new Neuroscience Gateway (NSG) registered 100+ users who used 250,000 CPU hours in only a few months.

Gateways, however, need to keep operational costs low and can often make use of common components. Science Gateway Platform as a Service (Sci-GaP) delivers middleware as a hosted, scalable third-party service while domain developers focus on user interfaces and domain-specific data in the gateway. The middleware handles things like authentication, application installation and reliable execution and help desk support.

One key to Sci-GaP is its openness. While based on the Apache Airavata project and the CIPRES Workbench Framework, community contributions are encouraged because of its open source, open

governance and open operations policies. The goal is robust, sustainable infrastructure with a cycle of development that improves reliability and prioritizes stakeholder requirements. The project is leveraging Internet2's Net+ mechanisms for converting SciGaP and its gateways into commodity services.

Zentner et al. [74] describes experiences and challenges with the large nanoHUB.org community. The authors define community as a "body of persons of common and especially professional interests scattered through a larger society." This makes support challenging because of the diversity of viewpoints and needs. The group constantly examines its policies to determine whether they are indirectly alienating part of the community or encouraging particular types of use.

nanoHUB's 10-year history with over 260,000 users annually provides a lot of data to analyze. 4000 resources contributed by 1000 authors. nanoHUB serves both the research and education community and the contribution model allows researchers to get their codes out into the community and in use in education very rapidly. The primary software challenges are twofold - support for the HUBzero framework and challenges related to the software contributed by the community.

The group has learned that community contributions are maximized with a tolerant licensing approach. HUBzero uses an LGPLv3 license so contributors can create unique components and license as they choose. If they make changes to source code, the original license must be maintained for redistribution. As far as contributed resources, these must be open access, but not necessarily open source. This allows contributors to meet the requirements of their institutions and funding agencies. Quality is maintained via user ratings. Documentation is encouraged and nanoHUB supplies regression test capabilities, but the user community provides ratings, poses questions and contributes to wishlists and citation counts - all of which incentivize code authors.

Terrel [75] describes support for the Python scientific community through two major efforts - the SciPy conference and the NumFOCUS foundation. Reliance on software in science has driven a huge demand for development, but this development is typically done as a side effort and often in a rush to publish without documentation and testing. While created by academics, software support often falls to industrial institutions. SciPy brings together industry, government, and academics to share their code and experience in an open environment.

NumFOCUS promotes open, usable scientific software while sustaining the community. Specific activities include educational programs, collaborative research tools and documentation and promotion of high-level languages, reproducible scientific research, and open-code development. Governance of the non-profit is a loose grantor-grantee relationship with projects allowing for monies to be placed in the groups accounts. This has raised money to hire developers for open code development, maintain testing machines, organize the PyData conference series, and sponsor community members to attend conferences.

Software sustainability relies on contributions from all sectors of the user community. Together SciPy and NumFOCUS support these sectors. By maximizing contributions growing the user base they help develop and mature Python.

Löffler et al. [61] describes the Cactus project which was started in 1996 by participants in USA Binary Black Hole Alliance Challenge. Cactus has a flesh and thorns, core and module, model - a community-oriented framework that allows researchers to easily work together with reusable and extendable software elements. Modules are compiled into an executable and can remain dormant, becoming active only when parameters and simulation data dictate. Users can use modules written by others or can write their own modules without changing other code. The community has grown and diversified beyond the original science areas.

The paper points out 4 keys to sustaining the community - modular design, growing a collaborative community, career paths and credit. In modular design, the Cactus project went far beyond standard practices of APIs. Domain specific languages (DSLs) allow decoupling of components - for example I/O, mesh refinement, PAPI counters, and boundary conditions abstracted from science code. In academia, publications are the main currency of credit. Because the project connects code developments to science, the work is publishable and modules are citable. Because of the open source, modular approach, programmers can see the impact of their contributions and often continue

work after graduation. Career paths remain a challenge, however. Tasks that are essential from a software engineering perspective are often not rewarded in academia. The best programmers in a science environment often have multidisciplinary expertise. That also is not rewarded in academia.

Wilkins-Diehr et al. [76] describes an NSF software institute effort to build a community of those creating science gateways or web portals for science. These gateways, as described in some of the other papers in this section, can be quite capable, having a remarkable impact on some parts of science.

This paper mentioned challenges highlighted by other papers in this section - mainly the conflict between funding for research vs infrastructure and the challenges around getting academic credit for infrastructure. Because the authors have studied many projects, they've also observed how development is often done in an isolated hobbyist environment. Developers are unable to take advantage of similar work done by others, isolation even when projects have common needs. But often projects struggle for sustainable funding because they provide infrastructure to conduct research and many times only the research is funded. Gateways also may start as small group research project, taking off in popularity once they go live without any long term plans for sustainability or without resources in the project to plan for such. Subsequent disruptions in service can limit effectiveness and test the limits of the research community's trust. The impact of gateways can be increased substantially if we understand what makes them successful.

Recommendations from an early study of successful gateways include the following. Leadership and management teams should design governance to represent multiple strengths and perspectives, plan for change and turnover in the future, recruit a development team that understands both the technical and domain-related issues, consider sustainability and measure success early and often. Successful projects recognize the benefits and costs of hiring a team of professionals, demonstrate credibility through stability and clarity of purpose, leverage the work of others and plan for flexibility. Successful projects identify an existing community and understand the communities' needs before beginning. Projects adapt as the communities' needs evolve. For funders, the lifecycle of technology projects must be considered. Solicitations should be designed to reward effective planning, recognize the benefits and limitations of both technology innovation and reuse, expect adjustments during the production process, copy effective models from other industries and sectors, and encourage partnerships that support gateway sustainability.

Through a business incubator type approach, the institute plans to provide a variety of services that could be shared amongst projects. Consultation and resources on topics such as business plan development, software engineering practices, software licensing options, usability, security and project management as well as a software repository and hosting service will be available. Experts will be available for multi-month assignments to help research teams build their own gateways, teaching them how to maintain and add to the work after the collaboration ends. Forums, symposia and an annual conference will connect members of the development community. A modular, layered framework that supports community contributions and allows developers to choose components will be delivered and finally workforce development activities will help train the next generation for careers in this cross- disciplinary area and build pools of institutional expertise that many projects can leverage. Shared services and forums can dramatically reduce the cost of building and sustaining gateways. Workforce development can encourage technology professionals to remain in the sciences.

Papers.

Communities Part 1 Papers.

- Reagan Moore. Extensible Generic Data Management Software [77]
- Karen Cranston, Todd Vision, Brian O'Meara, Hilmar Lapp. A grassroots approach to software sustainability [70]
- J.-L. Vay, C. G. R. Geddes, A. Koniges, A. Friedman, D. P. Grote, D. L. Bruhwiler. White Paper on DOE-HEP Accelerator Modeling Science Activities [67]

- Ketan Maheshwari, David Kelly, Scott J. Krieder, Justin M. Wozniak, Daniel S. Katz, Mei Zhi-Gang, Mainak Mookherjee. Reusability in Science: From Initial User Engagement to Dissemination of Results [68]
- Edmund Hart, Carl Boettiger, Karthik Ram, Scott Chamberlain. rOpenSci – a collaborative effort to develop R-based tools for facilitating Open Science [71]

Communities Part 2 Papers.

- L. Christopherson, R. Idaszak, S. Ahalt. Developing Scientific Software through the Open Community Engagement Process [72]
- Marlon Pierce, Suresh Marru, Mark A. Miller, Amit Majumdar, Borries Demeler. Science Gateway Operational Sustainability: Adopting a Platform-as-a-Service Approach [73]
- Lynn Zentner, Michael Zentner, Victoria Farnsworth, Michael McLennan, Krishna Madhavan, and Gerhard Klimeck, nanoHUB.org: Experiences and Challenges in Software Sustainability for a Large Scientific Community [74]
- Andy Terrel. Sustaining the Python Scientific Software Community [75]
- Frank Löffler, Steven R. Brandt, Gabrielle Allen and Erik Schnetter. Cactus: Issues for Sustainable Simulation Software [61]
- Nancy Wilkins-Diehr, Katherine Lawrence, Linda Hayden, Marlon Pierce, Suresh Marru, Michael McLennan, Michael Zentner, Rion Dooley, Dan Stanzione. Science Gateways and the Importance of Sustainability [76]

In addition, the following paper from another area was also discussed in this area.

- Marcus Hanwell, Amitha Perera, Wes Turner, Patrick O'Leary, Katie Osterdahl, Bill Hoffman, Will Schroeder. Sustainable Software Ecosystems for Open Science [69]

Industry & Economic Models.

- Anne C. Elster. Software for Science: Some Personal Reflections [78]
- Ian Foster, Vas Vasiliadis, Steven Tuecke. Software as a Service as a path to software sustainability [79]
- Marcus Hanwell, Amitha Perera, Wes Turner, Patrick O'Leary, Katie Osterdahl, Bill Hoffman, Will Schroeder. Sustainable Software Ecosystems for Open Science [69]

In addition, the following papers from other areas were also discussed in this area.

- Brian Blanton, Chris Lenhardt, A User Perspective on Sustainable Scientific Software [64]
- Markus Blatt. DUNE as an Example of Sustainable Open Source Scientific Software Development [31]
- Dhabaleswar K. Panda, Karen Tomko, Karl Schulz, Amitava Majumdar. The MVAPICH Project: Evolution and Sustainability of an Open Source Production Quality MPI Library for HPC [36]
- Andy Terrel. Sustaining the Python Scientific Software Community [75]

Education & Training.

- Ivan Girotto, Axel Kohlmeyer, David Grellscheid, Shawn T. Brown. Advanced Techniques for Scientific Programming and Collaborative Development of Open Source Software Packages at the International Centre for Theoretical Physics (ICTP) [80]
- Thomas Crawford. On the Development of Sustainable Software for Computational Chemistry [81]

In addition, the following papers from other areas were also discussed in this area.

- Charles R. Ferenbaugh, Experiments in Sustainable Software Practices for Future Architectures [27]
- David Koop, Juliana Freiere, Cláudio T. Silva, Enabling Reproducible Science with VisTrails [32]
- Sean Ahern, Eric Brugger, Brad Whitlock, Jeremy S. Meredith, Kathleen Biagas, Mark C. Miller, Hank Childs, VisIt: Experiences with Sustainable Software [33]

- Sou-Cheng (Terrya) Choi. MINRES-QLP Pack and Reliable Reproducible Research via Staunch Scientific Software [34]
- Frank Löffler, Steven R. Brandt, Gabrielle Allen and Erik Schnetter. Cactus: Issues for Sustainable Simulation Software [61]
- Erik Trainer, Chalalai Chaihirunkarn, James Herbsleb. The Big Effects of Short-term Efforts: A Catalyst for Community Engagement in Scientific Software [39]

6.3. **What are communities?**

6.4. **Challenges of community.** include metrics for community success (e.g., more external contributors than internal).

6.5. **Admirable scientific software communities.**

- example communities
- their (quick) origin stories.

6.6. **Resources for learning about software communities.**

- academic fields studying communities (and software communities)
- courses on online communities
- books
- need for software carpentry module on organizing communities?

7. Cross-cutting Issue: Defining Sustainability lead: Daniel S. Katz

So far this reads a lot like a collection of thoughts only, without "red line". Dan: assuming the previous note is from FL, what do you propose? The question of what was meant by "sustainability" in the context of software came up in many different parts of the workshop, specifically in the first keynote (§3.1), the Developing and Maintaining Software panel, and the Policy panel.

In the opening keynote, Philip Bourne suggested that perhaps sustainability can be defined as the effort that happens to make the essential things continue. This leads to having to decide what it is that we want to sustain, whether what we want to sustain is valuable, and finally, who would care if it went away, and how one measures how much they care.

In the Developing and Maintaining Software panel, there was some discussion on this topic: what does sustainability mean? It was pointed out that OSS Watch proposes a Software Sustainability Maturity Model to address the issue of how sustainability a particular element of software is, and says that this sustainability is important. "When choosing software for procurement or development reuse - regardless of the license and development model you will use - you need to consider the future. While a software product may satisfy today's needs, will it satisfy tomorrow's needs? Will the supplier still be around in five years' time? Will the supplier still care for all its customers in five years' time? Will the supplier be responsive to bug reports and feature requests? In other words, is the software sustainable?" [82]

Attendees suggested that a key question that the definition of sustainability is one issue on which the community needs to agree, and that ideally, an initial definition would be determined during the workshop, or at least some progress would be made towards this goal. Another topic that came up is what the goal of sustainability is. Perhaps it is reproducible science, or persistence, or quality, or something else. Similarly, some attendees want to understand how success in sustainability is measured. How does a group of developers know when they have actually achieved sustainable software? This led to a comment that sustainability should be addressed throughout the full software life cycle.

Another topic that came up during the Developing and Maintaining Software panel is the relationship of sustainability to other software attributes. Attendees asked "what is the relationship between sustainability and provenance?" And, "is usability separate from sustainability or a fundamental part of it?"

TABLE 1. Summary of Modeling Sustainability papers from Policy Panel. Adapted from [83].

Paper/Authors	Software	Sustainability	Approach to Understand or Evaluate Sustainability
Calero, et al. [48]	General notion of software. Not explicitly defined.	Sustainability is linked to quality.	Add to ISO
Venters, et al. [49]	Software as science software; increasingly complex; service-oriented computing	Extensibility, interoperability, maintainability, portability, reusability, scalability, efficiency	Use various architecture evaluation approaches to assess sustainability
Pierce, et al. [50]	Cyberinfrastructure software	Sustainable to the extent to which there is a community to support it	Open community governance
Katz and Proctor [51]	E-research infrastructures (i.e. cyberinfrastructure)	Persisting over time, meeting original needs and projected needs	Equates models for the creation of software with sustaining software
Lenhardt, et al. [52]	Broadly defined as software supporting science	Re-use; reproducible science	Comparing data management life cycle to software development life cycle
Weber, et al. [53]	Software broadly defined; a software ecosystem	Software niches	Ecological analysis and ecosystem

In addition, the Policy panel had a large amount of discussion about defining sustainability, as one of the subtopics in that panel was Modeling Sustainability, and modeling requires defining what will be modeled.

In this subtopic, two papers included discussion of the definition of sustainability. Venters et al. [49] mentioned that this is a rather ambiguous concept, and that the lack of an accepted definition gets in the way of integrating the concept into software engineering. They suggest that sustainability is a non-functional requirement, and that the quality of software architectures determines sustainability. They then propose that sustainability is a measure of a set of central quality attributes: extensibility, interoperability, maintainability, portability, reusability, and scalability. Finally, they develop an architecture evaluation framework based on scenarios that help to illuminate how to measure quality or sustainability at the architectural level.

Katz and Proctor [51] included a set of questions that could be used to measure software sustainability, and though these questions might falsely lead to yes or no answers, the complete set would determine a range of values for sustainability. These questions are:

- Will the software continue to provide the same functionality in the future, even if the environment (other parts of the infrastructure) changes?
- Is the functionality and usability clearly explained to new users?
- Do users have a mechanism to ask questions and to learn about the software?
- Will the functionality be correct in the future, even if the environment changes?
- Does it provide the functionality that existing and future users want?
- Does it incorporate new science/theory/tools as they develop?

Lenhardt [83] summarized the contributions of the Modeling Sustainability papers in the panel. As shown in Table 1, for each paper he discussed what software meant, whether there was a definition of sustainability, and what the approach was to either understand or evaluate sustainability.

Finally, in the workshop's closing session, one of the discussion topics was what success would look like for the set of WSSSPE activities beyond just the workshop. One of the answers that was suggested was having an agreed version of what we mean by sustainability.

8. CASE STUDIES LEAD: KETAN MAHESHWARI, CONTRIBUTORS: ANDY TERREL

Many important issues related to scientific software sustainability were discussed in the previous sections. In this section we discuss some of the software projects as case-studies to better understand the relevance of those issues and how are they connected to the software sustainability and related factors.

We classify the software projects discussed in the workshop in two broad categories. First, the utility software, comprising of software that enable and/or facilitate the development of other tools and techniques to carry out scientific work. This includes the software developed to efficiently utilize

new research infrastructures. Second, scientific software, comprising software that was developed with an aim to solve a specific scientific problem.

This classification is motivated by our argument that the software projects wildly vary in factors such as scope, purpose and usage. The development and management of software is significantly influenced by these factors. Consequently, the sustainability challenges faced by software differ and must be treated separately. For instance, the challenges faced by a gateway software development project such as *CIPRES* or *VisIT* are distinct to a niche software for ab initio modeling and simulation such as *VASP* or *Quantum Espresso*.

We aim to understand the practicalities of the key points discussed in the workshop, and hope to extract the best practices associated with the case studies. We discuss how the issues are applicable to the cases and try to understand how the course of current and future development projects might be altered to implement lessons learned. In particular, we discuss the following issues:

- How closely are the best practices discussed in the workshop were followed?
- Career path: Are the original developers still around?
- What conscious sustainability efforts were undertaken by stakeholders?
- What is the dominant sustainability factor: science, publication, (re)usability?

8.1. **Utility Software.** We classify software developed with a potentially wider audience and general purpose usage in mind as Utility software. In the context of research, a utility software could be defined as general purpose system applicable to one or many generic tasks and/or enabling other software to run by providing a suitable environment. Utility software typically do not address a fundamental research challenge in a given scientific domain. Examples are collaborative development frameworks such as *GitHub* and *Bitbucket*, distributed and workflow and generic computing frameworks such as *Galaxy*, *nanoHUB*, *SimGrid*, *Swift*, *Globus* and *VisTrails*, and visualization frameworks such as *VisIT*.

Development is often a high risk/reward undertaking and explores uncharted territories and is largely dominated by (re)usability factors. Owing to relatively large number of features, the development and prototype process is also lengthy which poses a significant survival risk. Ferenbaugh [27] further discusses the risks and challenges associated with software projects dealing with new architectures.

On the other hand, owing to its generic nature, utility software presents oppurtunities to be usable by a larger community. This makes it more visible which in turn leads to a broader and deeper participation. For instance, promoting collaborations across the breadth (e.g. different departments) and depth (e.g. stakeholders within a department) of community, one of the key ingredient of a sustainable process. Successful projects reap high rewards and have longer usage span. Development process becomes user-driven and self-sustaining.

One such example is the *Galaxy* project. It follows agile software development practices and implements standard practices such as test-driven development and socialized bug managing practices via *trello*. Galaxy *histories* and *toolshed* offers easy community sharing of data and tools further promoting a collaborative environment. The project very closely follows the guidelines described in [44] and many from [42]. Many utility software projects are often developed aiming better utilizing a particular, new infrastructure and architecture, e.g., *MVAPICH*, *VisIT*. Similarly, to leverage the power of emerging architectures such as CPU accelerators new code and libraries are required. The experience of one such effort as described in [27] met with a limited success but nonetheless with many invaluable lessons were learned about influential cultural and technical aspects in sustainable software development practices.

A relatively new paradigm in utility software is the software delivered as service over the web. With increasing popularity of cloud-based storage and computational environments, many users are leaning towards tools used as services. *GitHub* and *Bitbucket* can be arguably considered to be such tools, catering to collaborative development. For scientific users *Globus*-based tools are a case of service-based utility software discussed during the workshop. The data movement and sharing services offered by Globus can be easily used over the web by collaborating researchers.

8.2. **Scientific software.** Scientific software consists primarily of special-purpose software that was purpose-built for a target use-case scenario or fixed/frozen requirements in mind. Software projects pertaining to specific scientific domain often tend to be in a niche and the user community tends to be small to medium. They are mostly driven by the science and specific needs of a research group.

Long-term sustainability of scientific research software is often a significant challenge. Many submissions reported that software is practically considered a 'byproduct' of the actual research. Others contended that the software was not the main funded part of their research. Specific needs such as numerical accuracy and algorithmic optimization are some of the paramount requirements of most scientific software. The resulting codebase is relatively small. A smaller codebase and fixed requirements result in stability, ease of installation, and configurability. Many such projects mature and are treated as libraries to be used into larger systems such as some of the utility software discussed in the previous section. While the software can stay stable and require relatively low maintenance, the responsibility is often on the shoulders of a very few developers who might not be specialists in software development. Development tends to be linear and simplistic with a limited scope to follow software best practices.

Some examples of such software discussed as part of the workshop are, *DUNE*, *R/qtl*, *Kitware*, *PETSc*, *MINRES-QLP*, *FASTMath*. Most of the above efforts are focused on one scientific and/or applied mathematics domain. However, sometimes such projects grow beyond the initial vision of developers. One such example is *Kitware*, which while being a software product specializing in the scientific process, has a core focus of developing communities around software processes. An instance of this process is the development of the *CMake* build utility, which started out as a building tool for *ITK* but grew to become a generic build utility for C++ projects. Similarly, *PETSc* is growing towards becoming a general purpose utility system usable for solving variety of scientific problems.

In order to investigate the much discussed issue of career paths for developers of scientific software, we conducted a small survey of the use-cases discussed above with respect to the current status of its core developers. We looked at the workshop submissions and the original cited papers for the respective software. Where available, we looked at the webpages and source code of the earliest versions to find the original authors. We excluded relatively new projects such as *SciPy* and *numFOCUS* from this survey. we find that in a majority of scientific software more than one original developers are still around, in some cases for more than 10 years, (eg. *R/qtl*). Whereas, in the case of utility software projects, it appears that at least one original developer is still around often in a part-time role. In other cases, it appears that the original developers have went on to assume a larger role, again with the same project.

From this survey, it appears that the career paths for utility software developers are broader compared to the developers of niche software. However, as projects as a whole, the small-scale, specific software have a lower risk and cost in terms of sustaining as compared to utility software. Often the sustainability of such projects is achieved by the fact that the core developer or team heavily utilizes the software for their science, eg. *R/qtl, PETSc*. Furthermore, the development of scientific software requires more scientific skills compared with those of utility software. This diminishes the need of specialist programmers and practices in many cases where the bulk of development is carried out by a domain scientist.

Dan: I think there's some overlap between this and the discussion in the Developing and Support Software section. Perhaps this should go in the cross-cutting issues section? Let's write it down here - then we can decide if it goes elsewhere Ketan: yes, may be this section will split and merge into other sections

9. Conclusions

- [TODO: pull the discussion together]
- [TODO: add some analysis of the deficiencies and difficulties that are present in different fields, and those that are common?]
- [TODO: say something about licensing - lessons, advice, etc.?]

9.1. **Recommendations or Lessons.** if needed.

9.2. **Follow up actions.** or at least the discussion about them, and the current plans for future events.

conclusions from pre-workshop paper follows

The WSSSPE workshop has begun an experiment in how we can collaboratively build a workshop agenda. However, contributors also want to get credit for their participation in the process. And the workshop organizers want to make sure that the workshop content and their efforts are recorded. Ideally, there would be a service that would be able to index the contributions to the workshop, serving the authors, the organizers, and the larger community. But since there isn't such a service today, the workshop organizers are writing this initial report and making use of arXiv as a partial solution to provide a record of the workshop.

After the workshop, one or more additional papers will be created that will include the discussions at the workshop. These papers will likely have many authors, and may be submitted to peer-reviewed journals.

Acknowledgments

■ [TODO: anyone who needs to put something in here should]

Some of the work by Katz was supported by the National Science Foundation while working at the Foundation; any opinion, finding, 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.

Appendix A. List of attendees lead: Shel Swenson

The following is a partial list of attendees who were recorded on the Google doc [2] that was being used for live note taking at the workshop, or by the SC13 student volunteers, with some additions also made by the authors of this report.

Jay Alameda
Gabrielle Allen
David Andrs
Brian Austin
Lorena A. Barba
David Bernholdt
Phil Bourne
Karl Broman
Sharon Broude Geva
Jed Brown
Maxine Brown
David Bruhwiler
Bruno Bzeznik
Alexandru Calotoiu
Jeffrey Carver
Shreyas Cholia
Peng Huat Chua
Neil Chue Hong
John W. Cobb
Timothy Cockerill
Karen Cranston
Rion Dooley
Anshu Dubey
Marat Dukhan
Ian Foster
Juliana Freire
Jeffrey Frey
Derek Gaston
Allison Gehrke
Brian Glendenning
Christian Godenschwager
Derek Groen
Edmund Hart
Magne Haveraaen
Steven Heaton
Oscar Hernandez
James Hetherington
Simon Hettrick
Jonathan Hicks
Kenneth Hoste
James Howison
Daisie Huang
Shao-Ching Huang
Tsutomu Ikegami
Kaxuya Ishimura
Christian Iwainsky
Craig Jackson
Wesley Jones
Randall Judd
Shusuke Kasamatsu
Daniel S. Katz
Kerk Kee
Kellie Kercher
Mads Kristensen
Carolyn Lauzon
Arnaud Legrand
Chris Lenhardt
Michael Levy
Frank Löffler
Monica Lücke
Simon A. F. Lund
Arthur Maccabe
Paul Madden
Louis Maddox
Philip Maechling
Ketan Maheshwari
Brian Marker
Suresh Marru
Cezary Mazurek
James McClure
Matt McKenzie
Chris Mentzel

Paul Messina	Nicolas Renon	Andy Terrel
Mike Mikailov	Jason Riedy	George Thiruvathukal
J. Yates Monteith	Todd Rimer	Keren Tomko
Reagan More	Bill Sacks	John Towns
Rafael Morizawa	Andreas Schreiber	Erik Trainer
Pierre Neyron	William Scullin	Satori Tsuzuki
Lucas Nussbaum	Andrew Slaughter	Matthew Turk
Patrick O'Leary	Jaraslaw Slawinski	Eric van Wyk
Manish Parashar	Arfon Smith	Colin C. Venters
Cody Permann	Spencer Smith	Brice Videau
Jack Perdue	James Spencer	Tajendra Vir Singh
John Peterson	Eric Stahlberg	Von Welch
Quan Pham	Timothy Stitt	Nancy Wilkins-Diehr
Marlon Pierce	Hyoshin Sung	Theresa Windus
Heather Piwowar	Frédéric Suter	Felix Wolf
David Proctor	Shel Swenson	Rich Wolski
Sara Rambacher	Yoshio Tanaka	Lynn Zentner

References

[1] Daniel S. Katz, Gabrielle Allen, Neil Chue Hong, Manish Parashar, and David Proctor. First workshop on on sustainable software for science: Practice and experiences (WSSSPE): Submission and peer-review process, and results. Technical Report 1311.3523, arXiv, 2013. http://arxiv.org/abs/1311.3523.

[2] WSSSPE1 attendees. WSSSPE1 collaborative notes. https://docs.google.com/document/d/1eVfioGNlihXG_1Y8BgdCI6tXZKrybZgz5XuQHjT1oKU/. Accessed: 2014-02-03.

[3] National Science Foundation. A vision and strategy for software for science, engineering, and education: Cyberinfrastructure framework for the 21st century, 2012. NSF 12-113, http://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf12113.

[4] arXiv.org e-Print archive. http://arxiv.org. Accessed: 2014-02-03.

[5] figshare. http://figshare.com. Accessed: 2014-02-03.

[6] Philip E. Bourne. A recipe for sustainable software, 2013. A keynote presentation given at the First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1), November 17, Denver, Colorado, USA. Abstract available online: http://wssspe.researchcomputing.org.uk/wssspe1/keynotes/#bourne. Slides available online: www.slideshare.net/pebourne/a-recipe-for-sustainable-software.

[7] Public Library of Science. http://www.plos.org/. Accessed: 2014-02-11.

[8] Protein Data Bank. http://www.rcsb.org/pdb/. Accessed: 2014-02-05.

[9] GitHub. https://github.com/. Accessed: 2014-02-11.

[10] The BioJava Project. http://biojava.org. Accessed: 2014-02-07.

[11] Open Science Data Cloud. https://www.opensciencedatacloud.org/. Accessed: 2014-02-07.

[12] Stella Veretnik, J. Lynn Fink, and Philip E. Bourne. Computational biology resources lack persistence and usability. *PLoS Comput Biol*, 4(7):e1000136, 07 2008.

[13] Small Business Innovation Research. http://www.nsf.gov/eng/iip/sbir/. Accessed: 2014-02-07.

[14] National Science Foundation. Facilitation Awards for Scientists and Engineers with Disabilities. http://www.nsf.gov/pubs/policydocs/pappguide/nsf09_1/gpg_2.jsp#IID2.

[15] Rosemary Dickin. What does peer review mean when applied to computer code? http://blogs.plos.org/biologue/2013/08/08/what-does-peer-review-mean-when-applied-to-computer-code/. Accessed: 2014-02-07.

[16] Philip E. Bourne. Ten simple rules for getting ahead as a computational biologist in academia. *PLoS Comput Biol*, 7(1):e1002001, 01 2011.

[17] Arfon Smith. Scientific software and the open collaborative web, 2013. A keynote presentation given at the First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1), November 17, Denver, Colorado, USA. Abstract available online: http://wssspe.researchcomputing.org.uk/wssspe1/keynotes/#smith. Slides available online: http://is.gd/wssspe.

[18] Victoria Stodden. Why science is an open endeavor, 2013. A talk given at the Open Knowledge Conference, September 16–18, Geneva, Switzerland. Slides available online:http://www.stanford.edu/~vcs/talks/OKcon2013-Sept172013-STODDEN.pdf.

[19] C. Titus Brown. Laboratory of Genomics, Evolution and Development. http://ged.msu.edu. Accessed: 2014-02-07.

[20] RubyGems.org. http://rubygems.org/. Accessed: 2014-02-07.

[21] PyPI – The Python Package Index. https://pypi.python.org/pypi. Accessed: 2014-02-07.

[22] The Comprehensive Perl Archive Network (CPAN). http://www.cpan.org/. Accessed: 2014-02-07.
[23] Dan Foreman-Mackey and contributors. The Python ensemble sampling toolkit for affine-invariant MCMC. https://github.com/dfm/emcee. Accessed: 2014-02-07.
[24] Fernando Perez. An ambitious experiment in Data Science takes off: a biased, Open Source view from Berkeley. http://blog.fperez.org/2013/11/an-ambitious-experiment-in-data-science.html, 2013.
[25] Mark C. Miller, Lori Diachin, Satish Balay, Lois Curfman McInnes, and Barry Smith. Package management practices essential for interoperability: Lessons learned and strategies developed for fastmath. Technical Report 789055, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.789055.
[26] Karl W. Broman. Thirteen years of r/qtl: Just barely sustainable. Technical Report 1309.1192, arXiv, 2013. http://arxiv.org/abs/1309.1192.
[27] Charles R. Ferenbaugh. Experiments in sustainable software practices for future architectures. Technical Report 1309.1428, arXiv, 2013. http://arxiv.org/abs/1309.1428.
[28] Eric G. Stephan, Todd O. Elsethagen, Kerstin Kleese van Dam, and Laura Riihimaki. What comes first, the OWL or the bean? Technical Report 790738, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790738.
[29] Derek R. Gaston, John Peterson, Cody J. Permann, David Andrs, Andrew E. Slaughter, and Jason M. Miller. Continuous integration for concurrent computational framework and application development. Technical Report 790755, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790755.
[30] A. Dubey and B. Van Straalen. Experiences from software engineering of large scale AMR multiphysics code frameworks. Technical Report 1309.1781, arXiv, 2013. http://arxiv.org/abs/1309.1781.
[31] Makus Blatt. DUNE as an example of sustainable open source scientific software development. Technical Report 1309.1783, arXiv, 2013. http://arxiv.org/abs/1309.1783.
[32] David Koop, Juliana Freire, and Cláudio T. Silva. Enabling reproducible science with VisTrails. Technical Report 1309.1784, arXiv, 2013. http://arxiv.org/abs/1309.1784.
[33] Sean Ahern, Eric Brugger, Brad Whitlock, Jeremy S. Meredith, Kathleen Biagas, Mark C. Miller, and Hank Childs. VisIt: Experiences with sustainable software. Technical Report 1309.1796, arXiv, 2013. http://arxiv.org/abs/1309.1796.
[34] Sou-Cheng T. Choi. MINRES-QLP Pack and reliable reproducible research via staunch scientific software. Technical Report 791562, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791562.
[35] Michael Crusoe and C. Titus Brown. Walking the talk: adopting and adapting sustainable scientific software development processes in a small biology lab. Technical Report 791567, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791567.
[36] Dhabaleswar K. Panda, Karen Tomko, Karl Schulz, and Amitava Majumdar. The MVAPICH project: Evolution and sustainability of an open source production quality MPI library for HPC. Technical Report 791563, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791563.
[37] Eric M. Heien, Todd L. Miller, Becky Gietzel, and Louise H. Kellogg. Experiences with automated build and test for geodynamics simulation codes. Technical Report 1309.1199, arXiv, 2013. http://arxiv.org/abs/1309.1199.
[38] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. SimGrid: a sustained effort for the versatile simulation of large scale distributed systems. Technical Report 1309.1630, arXiv, 2013. http://arxiv.org/abs/1309.1630.
[39] Erik Trainer, Chalalai Chaihirunkarn, and James Herbsleb. The big effects of short-term efforts: A catalyst for community engagement in scientific software. Technical Report 790754, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790754.
[40] Jeremy Cohen, Chris Cantwell, Neil Chue Hong, David Moxey, Malcolm Illingworth, Andrew Turner, John Darlington, and Spencer Sherwin. Simplifying the development, use and sustainability of HPC software. Technical Report 1309.1101, arXiv, 2013. http://arxiv.org/abs/1309.1101.
[41] Jaroslaw Slawinski and Vaidy Sunderam. Towards semi-automatic deployment of scientific and engineering applications. Technical Report 791570, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791570.
[42] Andreas Prlić and James B. Procter. Ten simple rules for the open development of scientific software. *PLOS Computational Biology*, 8(12), 2012. http://dx.doi.org/10.1371/journal.pcbi.1002802.
[43] A. Dubey, S. Brandt, R. Brower, M. Giles, P. Hovland, D.Q. Lamb, F. Löffler, B. Norris, B. OShea, C. Rebbi, M. Snir, and R. Thakur. Software abstractions and methodologies for HPC simulation codes on future architectures. Technical Report 1309.1780, arXiv, 2013. http://arxiv.org/abs/1309.1780.
[44] Jeffrey C. Carver and George K. Thiruvathukal. Software engineering need not be difficult. Technical Report 830442, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.830442.
[45] Craig A. Stewart, Julie Wernert, Eric A. Wernert, William K. Barnett, and Von Welch. Initial findings from a study of best practices and models for cyberinfrastructure software sustainability. Technical Report 1309.1817, arXiv, 2013. http://arxiv.org/abs/1309.1817.
[46] Jed Brown, Matthew Knepley, and Barry Smith. Run-time extensibility: anything less is unsustainable. Technical Report 791571, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791571.

[47] Shel Swenson, Yogesh Simmhan, Viktor Prasanna, Manish Parashar, Jason Riedy, David Bader, and Richard Vuduc. Sustainable software development for next-gen sequencing (NGS) bioinformatics on emerging platforms. Technical Report 1309.1828, arXiv, 2013. http://arxiv.org/abs/1309.1828.

[48] Coral Calero, M.Angeles Moraga, and Manuel F. Bertoa. Towards a software product sustainability model. Technical Report 1309.1640, arXiv, 2013. http://arxiv.org/abs/1309.1640.

[49] Colin Venters, Lydia Lau, Michael K. Griffiths, Violeta Holmes, Rupert R. Ward, and Jie Xu. The blind men and the elephant: Towards a software sustainability architectural evaluation framework. Technical Report 790758, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790758.

[50] Marlon Pierce, Suresh Marru, and Chris Mattmann. Sustainable cyberinfrastructure software through open governance. Technical Report 790761, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790761.

[51] Daniel S. Katz and David Proctor. A framework for discussing e-research infrastructure sustainability. Technical Report 790767, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790767.

[52] Christopher Lenhardt, Stanley Ahalt, Brian Blanton, Laura Christopherson, and Ray Idaszak. Data management lifecycle and software lifecycle management in the context of conducting science. Technical Report 791561, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791561.

[53] Nicholas Weber, Andrea Thomer, and Michael Twidale. Niche modeling: Ecological metaphors for sustainable software in science. Technical Report 791564, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791564.

[54] Jason Priem and Heather Piwowar. Toward a comprehensive impact report for every software project. Technical Report 790651, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790651.

[55] Matthew G. Knepley, Jed Brown, Lois Curfman McInnes, and Barry Smith. Accurately citing software and algorithms used in publications. Technical Report 785731, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.785731.

[56] James Howison and James D. Herbsleb. Incentives and integration in scientific software production. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, page 459–470, San Antonio, TX, February 2013.

[57] James Howison and James D. Herbsleb. Scientific software production: incentives and collaboration. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, CSCW '11, page 513–522, Hangzhou, China, 2011. ACM.

[58] Matthew J. Bietz, Eric P. Baumer, and Charlotte P. Lee. Synergizing in cyberinfrastructure development. *Comput. Supported Coop. Work*, 19(3-4):245–281, August 2010.

[59] Neil Chue Hong, Brian Hole, and Samuel Moore. Software papers: improving the reusability and sustainability of scientific software. Technical Report 795303, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.795303.

[60] Daniel S. Katz. Citation and attribution of digital products: Social and technological concerns. Technical Report 791606, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791606.

[61] Frank Löffler, Steven R. Brandt, Gabrielle Allen, and Erik Schnetter. Cactus: Issues for sustainable simulation software. Technical Report 1309.1812, arXiv, 2013. http://arxiv.org/abs/1309.1812.

[62] Victoria Stodden and Sheila Miguez. Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. Technical Report 2322276, Social Science Research Network, 2013. http://dx.doi.org/10.2139/ssrn.2322276.

[63] Randy Heiland, Betsy Thomas, Von Welch, and Craig Jackson. Toward a research software security maturity model. Technical Report 1309.1677, arXiv, 2013. http://arxiv.org/abs/1309.1677.

[64] Brian Blanton and Christopher Lenhardt. A user perspective on sustainable scientific software. Technical Report 789028, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.789028.

[65] Daisie Huang and Hilmar Lapp. Software engineering as instrumentation for the long tail of scientific software. Technical Report 791560, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791560.

[66] Rich Wolski, Chandra Krintz, Hiranya Jayathilaka, Stratos Dimopoulos, and Alexander Pucher. Developing systems for API governance. Technical Report 790746, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790746.

[67] Jean-Luc Vay, Cameron G. R. Geddes, Alice Koniges, Alex Friedman, David P. Grote, and David Bruhwiler. White paper on DOE-HEP accelerator modeling science activities. Technical Report 793816, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.793816.

[68] Ketan Maheshwari, David Kelly, Scott J. Krieder, Justin M. Wozniak, Daniel S. Katz, Mei Zhi-Gang, and Mainak Mookherjee. Reusability in science: From initial user engagement to dissemination of results. Technical Report 1309.1813, arXiv, 2013. http://arxiv.org/abs/1309.1813.

[69] Marcus Hanwell, Amitha Perera, Wes Turner, Patrick O'Leary, Katie Osterdahl, Bill Hoffman, and Will Schroeder. Sustainable software ecosystems for open science. Technical Report 790756, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790756.

[70] Karen Cranston, Todd Vision, Brian O'Meara, and Hilmar Lapp. A grassroots approach to software sustainability. Technical Report 790739, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790739.

[71] Edmund Hart, Carl Boettiger, Karthik Ram, and Scott Chamberlain. ropensci - a collaborative effort to develop r-based tools for facilitating open science. Technical Report 791569, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791569.

[72] Laura Christopherson, Ray Idaszak, and Stan Ahalt. Developing scientific software through the open community engagement process. Technical Report 790723, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790723.

[73] Marlon Pierce, Suresh Marru, Borries Demeler, Amitava Majumdar, and Mark Miller. Science gateway operational sustainability: Adopting a platform-as-a-service approach. Technical Report 790760, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790760.

[74] Lynn Zentner, Michael Zentner, Victoria Farnsworth, Michael McLennan, Krishna Madhavan, and Gerhard Klimeck. nanoHUB.org: Experiences and challenges in software sustainability for a large scientific community. Technical Report 1309.1805, arXiv, 2013. http://arxiv.org/abs/1309.1805.

[75] Andy Terrel. Sustaining the Python scientific software community. Technical Report 791565, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791565.

[76] Nancy Wilkins-Diehr, Katherine Lawrence, Linda Hayden, Marlon Pierce, Suresh Marru, Michael McLennan, Michael Zentner, Rion Dooley, and Dan Stanzione. Science gateways and the importance of sustainability. Technical Report 790764, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790764.

[77] Reagan W. Moore. Extensible generic data management software. Technical Report 1309.5372, arXiv, 2013. http://arxiv.org/abs/1309.5372.

[78] Anne C. Elster. Software for science: Some personal reflections. Technical Report 1309.2357, arXiv, 2013. http://arxiv.org/abs/1309.2357.

[79] Ian Foster, Vas Vasiliadis, and Steven Tuecke. Software as a service as a path to software sustainability. Technical Report 791604, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.791604.

[80] Ivan Girotto, Axel Kohlmeyer, David Grellscheid, and Shawn T. Brown. Advanced techniques for scientific programming and collaborative development of open source software packages at the ICTP. Technical Report 796439, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.796439.

[81] T. Daniel Crawford. On the development of sustainable software for computational chemistry. Technical Report 790757, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.790757.

[82] Ross Gardler. Software sustainability maturity model. http://oss-watch.ac.uk/resources/ssmm. Accessed: 2014-02-06.

[83] Christopher Lenhardt. Summary of papers on science software sustainability models for WSSSPE panel II. Technical Report 853817, figshare, 2013. http://dx.doi.org/10.6084/m9.figshare.853817.