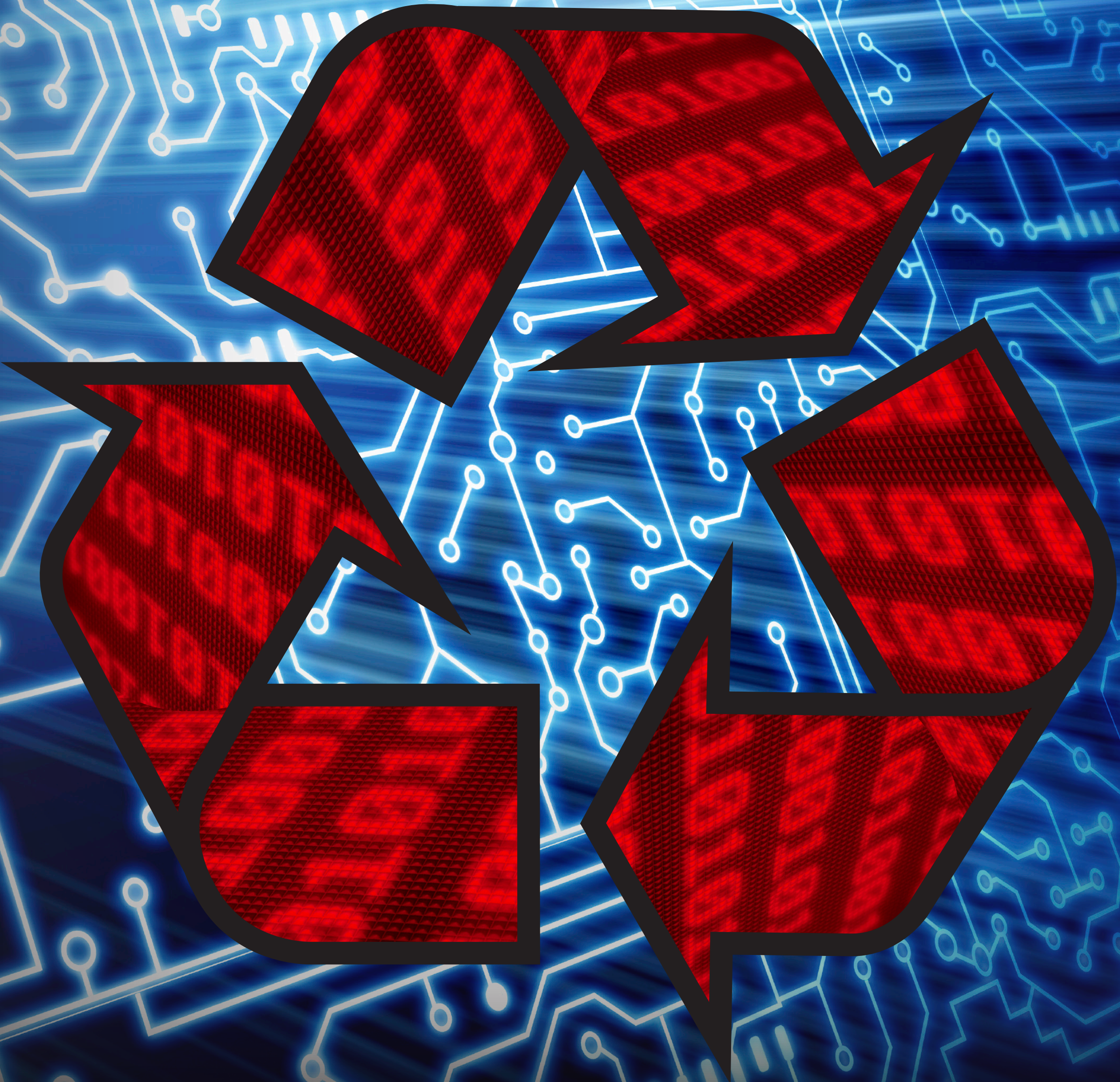
Cyberinfrastructure Software Sustainability and Reusability:

Report from an NSF-funded workshop
held 27 and 28 March 2009



Cyberinfrastructure Software Sustainability and Reusability:

Report from an NSF-funded workshop held 27 and 28 March 2009

Editors: Craig A. Stewart, Guy T. Almes, D. Scott McCaulay, and Bradley C. Wheeler

Contributing writers: Guy Almes[1], Amy Apon[2], Geoffrey Brown[3], Neil P. Chue Hong[4], David Lifka[5], Andrew Lumsdaine[6], Clifford Lynch[7], Marlon Pierce[8], Beth Plale[9], Ruth Pordes[10], Jennifer M. Schopf[11], Craig A. Stewart[12], Von Welch[13], Bradley C. Wheeler[14]

Please cite as: Stewart, Craig A., Guy T. Almes, and Bradley C. Wheeler (eds.). 2010. Cyberinfrastructure Software Sustainability and Reusability: Report from an NSF-funded workshop. Published by Indiana University, Bloomington, IN. Available from: http://hdl.handle.net/2022/6701

[1] galmes@tamu.edu / http://academy.tamu.edu/ / Texas A&M University / College Station, TX 77843
[2] aapon@uark.edu / http://www.uark.edu/home/ / University of Arkansas / Fayetteville, AR 72701
[3] geobrown@indiana.edu / http://www.cs.indiana.edu/ / Lindley Hall 330B / Indiana University / Bloomington, IN 47405
[4] N.ChueHong@omii.ac.uk / http://www.omii.ac.uk/ / Rm. 2409, JCMB, Mayfield Rd. | E: / Edinburgh, EH9 3JZ, UK
[5] lifka@cac.cornell.edu / http://www.cac.cornell.edu/ / Cornell Center for Advanced Computing / Frank H. T. Rhodes Hall / Hoy Road / Ithaca, NY 14853-3801
[6] lums@cs.indiana.edu / http://www.informatics.indiana.edu/ / Lindley Hall 301G / Indiana University / Bloomington, IN 47405
[7] cliff@cni.org / http://www.cni.org / Coalition for Networked Information / 21 Dupont Circle Ste 800 / Washington, DC 20036
[8] mpierce@cs.indiana.edu / http://pti.iu.edu/cgl/people/ / 2719 E 10th Street / Indiana University / Bloomington, IN 47408
[9] plale@cs.indiana.edu / http://www.cs.indiana.edu/ / Lindley Hall 215 / Indiana University / Bloomington, IN 47405
[10] ruth@fnal.gov / http://www.fnal.gov/ / Fermilab / P.O. Box 500 / Batavia, IL 60510
[11] jschopf@whoi.edu / http://www.whoi.edu/ / 266 Woods Hole Road, Woods Hole, MA 02543
[12] stewart@indiana.edu / http://ovpit.iu.edu/ / 601 Kirkwood Avenue, FH 116 / Bloomington, IN 47405-1223
[13] vwelch@ncsa.uiuc.edu / http://www.ncsa.illinois.edu/ / c/o NCSA/U. of Illinois / 1205 W. Clark / Urbana, IL 61801
[14] bwheeler@indiana.edu / http://ovpit.iu.edu/ / 601 Kirkwood Avenue, FH 116 / Bloomington, IN 47405-1223

Indiana University Pervasive Technology Institute
Indiana University
2719 East 10th Street Bloomington IN 47408
535 W. Michigan Street Indianapolis IN 46202

ISBN-13: 978-1456372897
ISBN-10: 1456372890

This material is based upon work supported in part by the National Science Foundation under Grant No. OCI-0829462. This material and the creation of this report are funded in part by a generous grant from the Lilly Endowment, Inc. to Indiana University to create the Pervasive Technology Institute.

Table of Contents

1. Executive Summary

The National Science Foundation's (NSF) strategy for 21st century innovation depends on creation of scientific and engineering software to enable discovery and innovation. The NSF workshop report "Planning for Cyberinfrastructure Support" states, "CI [Cyberinfrastructure] changes the rules and foundations of the research endeavor across much of NSF. CI software is a new class of artifact that should be the target of explicit design, construction, study, and evolution." [1] During March of 2009, Indiana University hosted an NSF-funded workshop on Cyberinfrastructure Software Sustainability and Reusability to examine the general issues of software sustainability and consider the question "given millions of dollars invested in software development, how will software important to the U.S. research and engineering communities be identified, maintained, and supported over years to decades?"

Cyberinfrastructure software is a critical national asset, and as the NSF pursues its strategies for 21st century innovation, there is no time better than the present to more thoroughly understand how to create and implement multiple sustainable models for software availability, usability, and provenance management. Sustainability in this case has two meanings: first, the need for long-term funding to develop and maintain the software and second, the need to create software that lives beyond the proposal by which its creation is funded.

Commercialization is not always a viable route to software sustainability. Alternatives such as open source and community source models for software creation and maintenance are proving viable for long-term sustainability of some types of applications. Some software critical to the U.S. national research agenda supports a relatively small community, and thus exists in a variety of less reliable states. When software developed as part of federally supported research develops a significant user base and takes on an important role in enabling discovery, it is vital to maintain and, where appropriate, extend such software. The lifecycle of such software should be conscientiously and actively planned and managed.

Sixty-five participants comprising a diverse and international group of experts attended the workshop *Cyberinfrastructure Software Sustainability and Reusability* to discuss these and other issues. Participants examined successful models for sustainability, such as the Open Science Grid and Sakai. Open source and community source software trends were examined, presenting an alternative to software commercialization. Open source software management, archival, and preservation solutions such as SourceForge and Apache were discussed in depth, along with the problems of sustainability for the repositories themselves. An examination of current successful models for sustainability of NSF-funded science and engineering software revealed that all develop at least some of their software as open source. Releasing software as open source is helpful in enabling sustainability, but not sufficient to ensure sustainability. (There are many other reasons to open-source software, including knowledge transfer, education, validation, and facilitating reuse.

Participants discussed development of cyberinfrastructure software as part of a research project and how it differs from professionally developed commercial software. Software may acquire status as infrastructure in at least two ways: someone writes something so useful that it becomes widely used within a community or across multiple communities; or software is developed as part of a plan to provide infrastructure. Recognizing and addressing the transition that occurs in the first of these cases, when existing software needs to be evolved into infrastructure, are particularly important and unaddressed needs.

Discussion of CI software focused on the realities of how it is developed today: often by students with little formal training in software development. It became clear that projects must be planned with sustainability and community needs in mind, rather than created as a means to a research end for a single researcher in a domain science. Considerable discussion throughout the workshop centered on the question, "How can educators assist in the production of sustainable, reusable software through better education in computer science and software engineering?"

The changing relationship between software sustainability and maintenance and the nature of scientific reproducibility informed the discussion on developing processes for life-cycle management of software, software artifacts, and data. A particularly important precursor to scientific reproducibility is the ability to collect and manage the provenance of both data and software, and more importantly, to manage the relationships between the two. It is also important that developers remember that hardware and user interfaces evolve rapidly. As much as possible, software should run independent of specific hardware and should use intuitive interfaces that do not depend upon extensive knowledge of the operating environment.

Commercialization is often recommended as one possible path to software sustainability in general. However, the overarching theme of this report is the need for the NSF to directly support some critical CI software as research infrastructure. Among these aspects are support for critical basic CI software functionalities, support for important CI codes that are foundational to other scientific research, and support for people who write, maintain, and provide consulting expertise about such software.

This document reflects the activities, discussions, and consensus of the two-day workshop and subsequent research and writing on specific points raised at the workshop. A two-day workshop is not a sufficient period of time to discuss, collect data, find appropriate references, and come to consensus on topics as complicated as those addressed during the course of the workshop. Therefore, after the workshop was concluded, a small group of volunteers formed a writing committee and pursued detailing discussions that took place and fleshing out analyses of topics that were brought up, following the spirit of discussion at the workshop. A penultimate version of this document was circulated among all attendees at the workshop, who were asked to endorse the document, suggest changes to the document, or object to the document in part or in whole. This generated suggestions that improved the precision of the document in important ways. More than two-thirds of the workshop attendees (other than representatives of federal funding agencies, who were recused from casting an opinion on the document) voiced their endorsement of the document, and none objected. With this outcome, it seems fair to assert that this document represents strong consensus of the workshop attendees.

Included in this document are 12 findings and 14 recommendations to the NSF:

Finding 1: A combination of focus on systematic collection and definition of user requirements and consistent application of good software engineering practices are important in enabling the sustained utility and sustainability of cyberinfrastructure software.

Finding 2: Because of the importance of reproducibility of scientific results and the challenges in maintaining software for small scientific communities over long periods of time, it is critical that cyberinfrastructure software be developed and released using an open source software license approved by the Open Source Initiative (http://www.opensource.org/), and that source code be managed and archived appropriately.

Finding 3: The NSF has generally been funding cyberinfrastructure software as if it were in the category of 'discovery'—funded through competitive peer-reviewed proposals based on the review criteria used for discovery proposals.

Finding 4: Cyberinfrastructure software is infrastructure—in particular, in terms of NSF strategic goal definition, cyberinfrastructure software is research infrastructure.

Finding 5: There is some cyberinfrastructure software that is uniquely or primarily valuable to the NSF and NSF-funded researchers, in the same fashion as other unique research infrastructure funded and sustained by the NSF. A particular piece of cyberinfrastructure software generally has a longer lifespan than cyberinfrastructure hardware.

> *Recommendation 1:* When funding software research and development, the NSF should put significant emphasis on the use of sound software engineering practices in evaluation of proposals and distribution of funding support.

> *Recommendation 2:* A condition of NSF support for creation and development of any cyberinfrastructure software should be the release of software under an open source license.

> *Recommendation 3:* Using the terminology of the National Science Foundation's four strategic goals outlined in its 2006–2011 strategic plan, the NSF should create funding mechanisms that support the ongoing sustainability and maintenance of cyberinfrastructure software as Research Infrastructure, employing mechanisms and evaluation criteria appropriate to Research Infrastructure rather than Discovery.

Finding 6: When cyberinfrastructure software is promoted as infrastructure (and/or there are requests to fund it as such), software should be measured and evaluated according to metrics relevant to it as infrastructure *per se*, such as the number of researchers who depend on use of such software. It is appropriate for the NSF to base decisions about whether or not to support the sustainability and maintenance of cyberinfrastructure software, and how much support to provide, on the basis of such metrics. Independent assessment of quality and impact should be strongly encouraged.

Finding 7: Much software critical to the work of NSF-funded researchers is not developed or maintained in a way that corresponds to its importance to the science and engineering community. The science and engineering research community of the U.S. suffers from the fact that software is not developed and maintained in a way that is more sustainable. Behavior change on the part of developers and funding agencies is required to make cyberinfrastructure software more sustainable.

Finding 8: Efforts such as those documented in the 2008 Computing Curriculum to emphasize software engineering education and the Defense Advanced Research Projects Agency (DARPA)-funded project to study computational science and engineering are of considerable help in establishing basic principles of software engineering methodology and techniques into computationally oriented scientific disciplines.

Finding 9: The computer science, computational science, and computationally oriented scientific disciplines would benefit from the widespread adoption of one or a very few standard excellent textbooks or other learning materials in software engineering.

> *Recommendation 4:* The NSF should support joint efforts with organizations such as the Association for Computing Machinery (ACM), the IEEE Computer Society, or Computing Research Association (CRA), incorporating the existing work done via DARPA support, to facilitate development of interdisciplinary courses and course materials on software engineering that are appropriate both for computational science and for engineering students who are not computer scientists.

> *Recommendation 5:* All researchers developing software should do so using good software engineering practices, even if they intend to use the software only for themselves, but particularly if there is reason to believe that the software might evolve into a longer-lived infrastructure role.

> *Recommendation 6:* The NSF should establish and fund processes for collecting community requirements and planning long-term CI software roadmaps to support community research objectives. Such a process may be informed by, but is distinct from, an enunciation of grand challenge problems. When the NSF solicits proposals for software to be developed intentionally as cyberinfrastructure, such solicitations should call for funding for intensive and extensive stakeholder requirements determination as part of funded activities.

Recommendation 7: The scientific community should promote scientific reproducibility. This can be done by requiring that: the provenance of software used in scientific research be carefully tracked and that versions used in particular experiments be documented in scientific publications; software, data, and software and data artifacts used in analyses should be available for review along with the text of a scientific publication as part of the peer-review process prior to publication; and data and software used in the development of a scientific publication should be escrowed or archived where they can be examined and re-verified when needed.

Recommendation 8: The NSF should require that data and software used in the development of a scientific publication based on NSF-funded research be escrowed or archived where it can be examined and re-verified as appropriate, in order to enable robust verification and reproducibility of scientific findings in the face of the cyberinfrastructure-dependent research environment of the 21st century.

Finding 10: Hardware emulation provides an excellent mechanism for enabling reuse of software after the hardware on which it was originally run no longer exists. Unresolved issues remain in terms of emulating special purpose processors, I/O generally, and network environments in particular.

Finding 11: User interfaces evolve rapidly over time and older interfaces are relatively quickly forgotten.

Recommendation 9: Sustainable software should be built with user interfaces that do not functionally depend upon the user having extensive knowledge of the software's operating environment.

Finding 12: In some cases, what researchers and practitioners care about having sustained is not a particular piece of software but rather a required capability.

Recommendation 10: The NSF should be prepared to make decisions to fund a succession of software, over time, that provide key required capabilities and in so doing focus on a limited number of robust codes maintaining a particular functionality at any given time.

Recommendation 11: The NSF should create a funding program to fund CI software development, hardening, support, and sustainability. Such a program might be based on long-term funding for "software centers" modeled after the existing Science and Technology Center program.

Recommendation 12: The U.S. research community should, when feasible, pursue the model of collaborating within the framework of a not-for-profit foundation as a way to maintain and sustain cyberinfrastructure software development and support.

Recommendation 13: The NSF should fund empirical studies of software sustainability efforts, so that the NSF, other funding agencies, and the science and engineering community generally can learn from and build upon real-world experience in developing and supporting cyberinfrastructure sustainably.

Recommendation 14: The NSF should encourage and support the flow of information between the global open source community, industry, and academia, focused particularly on encouraging extensible and interoperable CI development, through support for development and maintenance of software standards when appropriate.

2. Introduction

2.1. *Motivation and Background*

The National Science Foundation's (NSF) strategy for 21st century innovation depends explicitly on the creation of new software to enable scientific discovery. Indeed, the rapid creation of new software is essential for U.S. research competitiveness in today's global environment of intellectual competition. The 2005 NSF workshop report "Planning for Cyberinfrastructure Software" states, "CI [Cyberinfrastructure] changes the rules and foundations of the research endeavor across much of NSF. CI software is a new class of artifact that should be the target of explicit design, construction, study, and evolution" [1]. This 2005 report summarizes key research and makes several recommendations regarding the creation and coordination of cyberinfrastructure software. Similarly, the National Science Foundation Cyberinfrastructure Council document "Cyberinfrastructure Vision for 21st Century Discovery," published in 2007, discusses the critical importance of cyberinfrastructure software, including its hardening and maintenance [2].

"Cyberinfrastructure Vision for 21st Century Discovery" [2] includes an appendix listing dozens of workshops and reports related to cyberinfrastructure hardware and software. None addresses cyberinfrastructure software sustainability as its primary focus. At present, the U.S. is investing hundreds of millions of dollars annually in networking and hardware infrastructure, people, and software. The importance of software to scientific research in simulation and data analysis is clearly recognized [3]. The NSF has well-developed plans regarding computational and networking infrastructure and people. There is much less clarity on how to make the software component of this trio sustainable over time, even though this is critical to science and engineering research, development, and delivery.

There are two critical aspects to sustainability. One is sustaining the activity of software projects so that they can continue effectively, building functionality and supporting users over time. A second aspect of sustainability is based on the recognition that sooner or later activity on most software projects comes to an end. For scientific research done using such software to be reproducible, it must be possible to use the software after active work on the project has ended.

Considering the first aspect of sustainability, when software developed with support of the NSF or other federal agencies garners a significant user base and takes on an important role enabling science and engineering discovery, it is important to maintain such software as long as it remains valuable and, when appropriate, extend it. In this way, software can be refined and made more robust generally so as to empower current and future scientists to make new discoveries. If science performed through use of such software is to be reproducible, then it must be possible to reuse software over long periods of time.

Much prior discussion of sustainability has looked at commercialization and open source software as two primary approaches to creating sustainability for cyberinfrastructure software. Commercialization of software developed as a result of academic research has been for decades viewed as a beneficial vehicle for sustainability of technology generally. The Bayh-Dole Act [4] was created in 1980 with the goal of enhancing technology transfer from federally-funded research into the private sector. NSF programs such as SBIR (Small Business Innovation Research) and STTR (Small Business Technology Transfer) exist specifically to promote commercialization. Several reports since have discussed these efforts in general including commercialization of software (e.g., [5-7]). The NSF "Cyberinfrastructure Vision for 21st Century

Discovery" vision document speaks specifically of "facilitating the transition of commercially viable software into the private sector." The development of the Netscape web browser and subsequent transition of web browser technology to the private sector is perhaps the most dramatic example of the impact of cyberinfrastructure software commercialization. Other important scientific software applications developed initially within universities have been transitioned successfully into the private sector. Many examples of this sort of success in commercialization are found in areas of statistical and mathematical software (e.g., SPSS [8] and Maple [9]).

When commercialization is a viable route to software sustainability, there can be widespread benefits to the public research sector, private research sector, and U.S. economy. In such cases there are also benefits to the federal funding agencies that funded the initial phases of a software project but are not required to cover the ongoing costs of software maintenance as a result of commercialization. Commercialization is not a panacea, however. Courant and Griffiths [10] identified significant challenges faced by academic institutions in sustaining their use of commercial enterprise applications, including concerns about cost of commercialized software and the ceding of control from academia to the commercial sector. Also, particularly as regards cyberinfrastructure software, some software critical to the scientific endeavor will never be commercially viable. One common reason is that the audience for some critical cyberinfrastructure software is simply too small to support a commercial software effort. Another is that the combination of a requirement for a high rate of change and a high tolerance for low robustness in the software itself is not compatible with a commercial software effort.

Alternatives to commercialization, such as the open source movement, are emerging for long-term support of software. An excellent general overview of open source models is given in Weber [11]. The commercialization of open source software has been discussed by Karels [12]. Recent symposia such as the International Federation for Information Processing [13] testify to growing worldwide attention to this phenomenon. The Open Source Software Watch [14] provides an ongoing resource for open source development projects. Wheeler [15] has reviewed recent successes and challenges of university open source. Community source business models have been discussed by Perens [16], Gandel and Wheeler [17], Mann [18], and Valimaki [19]. The overarching theme to many of these articles is that while there is great merit in the open source and community source models, there are significant challenges to maintaining software over the long term through a strictly open source model.

Open source software of real value to the U.S. scientific community and important to the U.S. national research agenda generally exists in states ranging from "as persistent as the next grant supporting maintenance" to "supported by the persistence of a very small community of volunteers" to "exists only as code stored in SourceForge." As of the end of October 2009, SourceForge [20] included a total of 158,332 projects. Of these, 142,271, or 89.9 percent, were not updated in the 12 months prior, and 61.5 percent have not been updated in the past five years. In one sense, SourceForge is doing a tremendous job of housing open source projects and preserving the availability of open source software. However, SourceForge's success in preserving open source software creates a challenge for the scientist. Within such a large collection of software, it's challenging to find the right tools, discover easily which tools have robust support, or identify which tools enjoy the greatest support among the communities of practice and virtual organizations that use them.

The Apache Software Foundation [21] is a particular example of an open source software organization that attempts a more comprehensive management approach than repository services such as SourceForge. SourceForge's strength is the wide variety of tools and services that it provides to support open source software development. It, however, is not very discriminating in the projects that it accepts, and it does not enforce sunsetting or removal of inactive projects. The Apache Software Foundation, while providing some hosting capabilities and infrastructure, is primarily an entity for providing organizational support for open source software. The foundation has a board of directors, is a 501(c)(3) nonprofit organization, and of course has developed the well-known Apache licenses. By requiring its member projects to meet and uphold certain organizational and quality standards, the foundation has been able to promote the "Apache" brand: it is a desirable (and obtainable) goal for an open source effort to become a full-fledged Apache project. The Apache Software Foundation considers the full life cycle of software; that is, it also has policies for handling inactive, retired, and discontinued projects, which are moved to its graveyard. Projects moved to the graveyard are preserved and may be reactivated, but they are otherwise clearly distinguished from the active projects.

The Linux operating system is maintained and sustained by a combination of open source community effort and commercial support through companies such as RedHat and Novell. Other open source tools are managed by large hardware corporations who see support of open source tools as a part of their corporate strategy.

Wheeler [15, 22] reviewed trends in open source software and draws out several conclusions. One is that economies of scale, self-interest of universities, and effective collaborations can develop, deliver, and support in the short-to-medium term a wide variety of software useful in higher education. Wheeler outlines a taxonomy of possible future scenarios for sustainability of software within higher education. The most effective and long-standing case examples, however, are those tools that are of broad enough interest to be valuable to many universities in terms of their institutional function, such as course management and university financial systems. For example, the Sakai Foundation [23], which collaboratively produces a highly successful course management system, includes as of 2009 a total of 68 academic members and 13 industrial partners, and is used by more than 160 academic institutions. The Kuali Foundation [24], which is developing business process and financial software for universities, includes a total of 33 academic members and 10 industrial partners. Both are organized as 501(c)(3) educational nonprofit organizations.

There are two examples of successful use of a foundation approach to scientific software. One is the R foundation for Statistical Computing [25], which sustains the R statistical software package and is supported by 32 institutions and dozens of individuals. Another is the HUBzero Consortium [26], a new initiative that aims to generalize, expand, and sustain the very successful nanoHUB software [27, 28]. The HUBzero consortium is creating a platform that supports use of applications, access to information, and collaboration for scientific and educational activities. However, the types of advanced cyberinfrastructure tools on which the most advanced research in the U.S. depends are not likely a good match for the models of success identified by Wheeler.

In addition to the question of how to manage long-term sustainability of CI software of tremendous yet narrow scientific importance, there are significant problems as regards the related question of reusability and reproducibility: "ten years in the future, how do I make software program 'x' run?" At a minimum this implies executing the software on well-defined inputs. Doing this requires either maintaining hardware platforms (in general not a viable strategy) or providing hardware emulation frameworks. Replicating the use of a particular piece of software implies the ability to replicate everything that software depends upon, ranging from libraries and operating systems to compilers. This raises the practical question of identifying software dependencies—an area in which RedHat and Debian have made excellent starts with their package management software. This does not address the more difficult problem of what might be called "environmental dependencies" (e.g., databases that a package assumes are accessible). There is a very extensive body of research, largely from the library, digital library, and archival communities dealing with digital preservation, provenance, environmental dependencies, and related matters, although much of the work has been focused on documents (and by extension the software used to render them) rather than the broader area of applications and infrastructure software preservation. The Coalition for Networked Information [29] and research disseminated through the Council on Library and Information Resources [30] maintain information covering much of this work on their web sites; another important resource is the UK Digital Curation Center hosted at the University of Edinburgh [31]. Still, how a researcher in the future might reproduce scientific computations done today remains an important problem not yet generally solved.

There are then three existing models for sustainability while software is actively used and serving an important role in the ongoing scientific discovery enterprise—commercialization, open source, and community source, none of which seems to be an ideal solution for the problem of sustaining cyberinfrastructure software. Commercialization may be a solution for a narrow set of software. Production and use of open source software may be an element of a solution in some cases, but simply saying "make it open source" is not a solution. Community and foundation-based approaches are interesting but yet without much track record overall. Given important questions regarding cyberinfrastructure sustainability and no obvious answers, the workshop on *Cyberinfrastructure Software Sustainability and Reusability* was proposed to and funded by the National Science Foundation to consider these problems.

2.2. ***Preparation for and Execution of the Workshop***

Input for the workshop on *Cyberinfrastructure Software Sustainability and Reusability* was solicited in advance. Brief position papers were solicited from the cyberinfrastructure community. A total of 20 papers were submitted. These position papers are included as Appendix 1 of this report. Appendix 2 is a collection of all the specific recommendations made in these submitted position papers.

The workshop *Cyberinfrastructure Software Sustainability and Reusability* was held March 26 and 27, 2009, at the University Place Convention Center on the Indiana University–Purdue University Indianapolis campus. A total of 65 participants attended, along with six Indiana University staff supporting the workshop. The participants included two representatives of the NSF.

Workshop participants were invited from a diverse and international set of experts. In addition, several individuals and representatives of organizations were invited to participate in this workshop as a result of position paper submissions. Appendix 3 of this report is a list of the attendees. Appendix 4 is the workshop program. Appendix 5 includes the presentations of invited plenary speakers.

This document reflects the activities, discussions, and consensus of the two-day workshop and subsequent research and writing on specific points suggested at the workshop. A two-day workshop is not a sufficient period of time to discuss, collect data, find appropriate references, and come to consensus on topics as complicated as those addressed during the course of the workshop. Therefore, after the workshop was concluded, a small group of volunteers formed a writing committee and pursued detailing discussions that were had and fleshing out analyses of topics that were brought up, following the spirit of discussion at the workshop. A penultimate version of this document was circulated among all attendees at the workshop, who were asked to endorse the document, suggest changes to the document, or object to the document in part or in whole. This generated suggestions that improved the precision of the document in important ways. (There were a small number of representatives from federal agencies present at this workshop. Some of them made suggestions regarding faculty matters of this report and aligning the report to the actual workshop activities. The representatives of federal agencies were recused from offering a formal opinion on the document itself so as to not create a conflict of interest or any appearance thereof). In the end more than two-thirds of the workshop attendees voiced their endorsement of the document, and none objected. With this outcome, it seems fair to claim this document represents strong consensus of the workshop attendees.

3. **Definitions and Existing Models**

For purposes of this workshop report, we adopt the definition of cyberinfrastructure created for an earlier workshop conducted jointly by the Coalition for Academic Scientific Computation and the EDUCAUSE Campus Cyberinfrastructure committee [32] (this definition based on one developed earlier at Indiana University [33]).

> Cyberinfrastructure consists of computational systems, data and information management, advanced instruments, visualization environments, and people, all linked together by software and advanced networks to improve scholarly productivity and enable knowledge breakthroughs and discoveries not otherwise possible.

There is a particular sense in the use of the term cyberinfrastructure that is much more narrow than one might attach to "information technology infrastructure" or "computer infrastructure." One of the reasons that definitions in computer and computational science are difficult, however, is that software may be put to many different uses. Webster's dictionary offers the following definition of infrastructure:

The first use of the term cyberinfrastructure seems to have been in 1998 in a press briefing by Richard Clarke, then National Coordinator for Security, Infrastructure Protection, and Counter-Terrorism; and Jeffrey Hunker, Director of the Critical Infrastructure Assurance Office [35]. The use of this term really took off, however, after its inclusion in the 2003 "Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure" (generally referred to as the Atkins Report) [36]. The sense of the definition is that of a suite of resources that work together to enable discovery that is not achievable at any given time by typical information technology infrastructure (recognizing that today's cutting-edge capabilities will appear routine at some point in the future).

Cyberinfrastructure software is the software that enables cyberinfrastructure to function. Like cyberinfrastructure itself, there is not a precise definition. The focus of attention for this workshop centered on grid middleware and other tools that are at the core of enabling the function of the national NSF-funded cyberinfrastructure. Middleware (defined as "A communications layer that allows applications to interact across hardware and network environments" [37]), particularly grid middleware and grid workflow management tools, are at the center of the spectrum of what could be considered

---

in·fra·struc·ture [34]

1 : the underlying foundation or basic framework (as of a system or organization)

2 : the permanent installations required for military purposes

3 : the system of public works of a country, state, or region; *also*: the resources (as personnel, buildings, or equipment) required for an activity

cyberinfrastructure software. Recommendations made in this document regarding NSF funding for cyberinfrastructure software speak specifically and primarily about this type of software.

To try to put some bounds on the concept as viewed by workshop attendees, operating systems at the base of the software stack are not cyberinfrastructure software. Operating systems are a very special case, and their sustainability is a focus of intense, widespread, and largely successful attention. There are a number of software applications that clearly are cyberinfrastructure software—software in this category includes software such as Condor [38], the globus toolkit [39], Genesis II [40], MPI library implementations (MPICH [41] and Open MPI [42]), CACTUS [43], Pegasus [44], and data-centric tools such as the Storage Resource Broker (SRB) [45] and integrated Rule Oriented Data Systems (iRODS) [46]. The many software tools that create the middleware stack used by the Open Science Grid certainly constitute cyberinfrastructure software. Math libraries are a special case, but it seems reasonable to consider BLAS libraries (such as GotoBLAS [47]) and fast Fourier Transforms (such as FFTW [48]) as cyberinfrastructure software. Repository software such as Dspace [49], Fedora [50], and Eprints [51] could also reasonably be considered CI software. Software such as Sakai or R is borderline; one could argue that in many circumstances they are indeed cyberinfrastructure software. Programs such as Microsoft Word [52] and WordPerfect [53] are clearly end-user applications, not cyberinfrastructure software. Due to the focus of this workshop, there is relatively little attention in this report given specifically to science codes. Indeed, a subsequent workshop on science application codes may well be warranted.

To the extent that software developers wish to have their software viewed as cyberinfrastructure software, and receive funding on that basis, the metrics and evaluation criteria set forth in this document should be useful in justifying support for software as cyberinfrastructure software. Suggestions regarding education for sustainable software and development of sustainable software contained herein should be useful to any and all who develop software and all who educate future software developers of all sorts.

3.1. *Definitions of Sustainability*

Webster's dictionary offers the following definitions of sustain and sustainability:

sus·tain [54]	1 : to give support or relief to
	2 : to supply with sustenance : nourish
	3 : keep up, prolong
	4 : to support the weight of : prop; also : to carry or withstand (a weight or pressure)
	5 : to buoy up <sustained by hope>
	6 a : to bear up under b : suffer, undergo <sustained heavy losses>
	7 a : to support as true, legal, or just b : to allow or admit as valid <the court sustained the motion>
	8 : to support by adequate proof : confirm <testimony that sustains our contention>
sus·tain·able [55]	1 : capable of being sustained
	2 a : of, relating to, or being a method of harvesting or using a resource so that the resource is not depleted or permanently damaged <sustainable techniques> <sustainable agriculture> b : of or relating to a lifestyle involving the use of sustainable methods <sustainable society>

In terms of cyberinfrastructure software, there are several different sorts of sustainability and usability that one might contemplate, as follows:

- Saying that a piece of cyberinfrastructure software should be sustainable means that it should be well maintained and kept up to date (e.g., compiled, tested, and distributed for current versions of operating systems).
- Saying that a piece of cyberinfrastructure software should be reusable means that reasonably trained scientists should be able to discover the software, find the information and other software upon which that software is dependent so as to run and use it for scientific research, and have access to test suites that validate that the software is performing as intended by the authors of the version one is using. (This corresponds to one of the definitions of sustainability presented by Jennifer Schopf in her talk—sustainability as the "ability to maintain a certain process or state.")

In a software context, one may reasonably draw a tight linkage between sustainability and reusability, in the sense that one aspect of sustainability is sustainability of the value of code elements and modules through reusability—the capability to reuse components, leveraging the initial effort in the creation of CI software and sustaining the realization of its value.

3.2. *Existing Models for Sustainability*

A brief taxonomy of sustainability models is presented below, modified slightly from a taxonomy developed by Hong, presented at a JISC-sponsored workshop on Models of Sustainability held on 3–4 December 2007 [56] and reported on in [57].

- Open source variants
  - Developers are unpaid volunteers.
  - Developers are compensated by some combination of paid-for maintenance, book publications, speaking, and consulting.
  - Dual version open source
    - A free version is developed with limited functionality.
    - A commercial version is developed with richer functionality.
  - Development is sustained by an institution or corporate supporters.
- Community source code sustained by a foundation
  - Developers are mostly paid by universities that contribute their time in lieu of alternate costs for commercial licensing and maintenance, and a very few foundation staff help coordinate distributed design, coding, quality assurance, and release packaging work by distributed community participants.
- Funding agency support
  - A software project is sustained by a variety of interlocking and intermingled grants—the "grant mosaic" model of federal funding.
  - "Flagship codes" are so important that it is appropriate to have ongoing federal funding for a particular major software application or software suite.
  - Funding of specific work is built into a project budget, and based on user needs and distribution of tokens—e.g., HECToR CSE [58].
- Commercial software

4. Exemplars of Success in Cyberinfrastructure Software Sustainability

As ways to ground discussion of software sustainability in real experience, participants at the Cyberinfrastructure Software Sustainability and Reusability workshop discussed several exemplars of success in software sustainability. Among the exemplars discussed were the software used by the Open Science Grid (OSG), MyProxy, and Sakai, which are described in greater detail below. These three projects represent three different sustainability models as discussed in Section 3.2. OSG is most akin to a group of user communities who have collectively selected a set of software of value to them and is sustaining that set via support from supporting funding agencies. MyProxy represents support by a development team under the "grant mosaic" model. Finally, Sakai represents a community code sustained by a foundation and used by a group of universities and colleges, all of whom have interests and needs largely in common. (Other good examples of software sustainability successes are described in some of the position papers included in Appendix 1, such as the description of the sustainability and transition from SRB to iRODS included in the position paper by Reagan Moore, Arcot Rajasekar, Mike Wan, and Wayne Schroeder.)

4.1. *Open Science Grid*

The Open Science Grid Consortium [59] provides a multidisciplinary collaboration in support of the computing needs of long-lived physics, as well as other domain science communities across the Department of Energy (DOE) and NSF. The Open Science Grid (OSG) builds, tests, documents, distributes, and supports more than 60 software packages for its users. The requests and requirements of the user communities determine what software is included. Their lack of use of and need for the software determines when software is removed. The Open Science Grid Consortium thus sustains an entire software suite, rather than one or a few particular applications or codes.

The OSG strives to not do software development outside of utilities to support the distribution, installation, and configuration of the integrated collections of software components needed by the users. The software modules are developed by external groups and include common open source software (e.g., Apache [21]), computer science/engineering developed middleware (e.g., Condor [38], GUMS [60]), and peer infrastructure developed software (e.g., EGEE gLite [61]).

The OSG Consortium separates the software into two layers:

- The first is the Virtual Data Toolkit (VDT) [62] and is system-agnostic. That is, it does not contain specific configuration required to be part of a specific distributed system (community system, campus or national grid).
- The second layer is the OSG software stack [63], which adds configuration information specific to OSG, such as the specific Certificate Authorities trusted by the OSG Consortium.

If there is software deemed useful to include in the OSG software stack and it needs a small amount of extension in its functionality, robustness, or scalability, the OSG may inject short-term effort to help the software provider do this work.

The OSG Consortium follows the process below for the life cycle of a software component:

- Collect needs and capability requests from user communities, including prioritizing and assessing the common needs.

- Validate the software for robustness, completeness, documentation, and impact on existing software modules.
- Assess the Consortium's ability to integrate, test, and provide front-line support for the software.
- Include the software in a (development and then production) release of the software stack. This results in the software becoming part of the build, test, integration, and documentation environments.
- Plan for a security audit of the software.
- Revisit the use of and need for the software at each major release of the software stack to understand whether it is approaching end-of-life.

Based on the OSG's experience, they have made several design decisions about the software stack:

- *Binary distributions.* The OSG Consortium has found that building all of the different software is both complex and slow, so they prefer to do this on behalf of users. While this can be performed on a user's computer, it is complex and error-prone.
- *Configuration tools.* The OSG Consortium provides tools for configuring the software to work together in specific scenarios. For example, they provide configuration tools and files to install specific web applications into Tomcat, to connect Tomcat with Apache, and to configure specific security settings across Tomcat and Apache.
- *Packaging of Installation Sets.* The OSG Consortium is transitioning to packaging and distribution based on native packages such as RPM Package Manager. This will better match the user needs for ease and commonality of installation and configuration with the other software they use. To date they have been using Pacman [64], a user-level tool that supports non-root installations, multiple installations, and support for a hierarchy of software caches for different subsets of the available packages.
- *Release Management.* The OSG Consortium follows a process of well-defined production releases of the software. Site administrators and user communities can take a while to upgrade to new versions of the software, especially if they are in the middle of a long production run. The OSG Consortium maintains support for older releases as needed by the users—though they prefer to support only two major releases at any time. The OSG Consortium supports a security patch and "minor bug" release and installation process for fixes that must be delivered quickly. In these cases, they rely on less testing. The OSG Consortium assesses the risk of reduced testing versus not getting the patch out and installed, and adjusts the testing accordingly.
- *Integrated Build and Testing.* The OSG Consortium has a rich system for integrated build and testing of the software to better ensure that the solutions provided by the software will perform well in terms of performance and impact on the systems on which they run, and for the users they support. Each software component is built for all the platforms they support. The OSG Consortium relies on the NSF Middleware Initiative (NMI) Build and Test Facility at the University of Wisconsin–Madison [65] and the Metronome software [66] to manage builds. To support the goal of smoothly integrating software for users, the Consortium sometimes patches the software to provide bug fixes or features needed by OSG stakeholders.
- *Integrated System Testing.* This provides end-user testing in scenarios that are as close to production as possible. The OSG Integration Testbed (ITB) is a grid of 10–20 sites. Each site donates a small number of computers and storage to provide a production-like environment to run grid jobs. End users who have scientific applications that need to be run do the testing. This process helps fix problems before the software is released to production, gives confidence to users that the software really works, and assures virtual organizations (VOs) and production grid sites that it is safe to upgrade to the new version. This involvement from the community in testing the software has proven to be invaluable.
- *Open source is a requirement.* The OSG Consortium uses only software that is open source and buildable from source on their systems. This is necessary to reduce the risk if the software developer will no longer support the package. If the software is still in use by OSG

communities, they follow a process to find effort to support it either internally or as a contribution from the members of the OSG Consortium.

Key lessons: The Open Science Grid Consortium maintains a software stack based on use of Open Source codes. The components of the software stack are selected and integrated on the basis of the common needs of more than one user community (and virtual organization). The OSG Consortium minimizes the extent to which they develop software, and has an extensive software integration, testing, and distribution system. The OSG Consortium uses only open source software and is prepared to support and maintain such software if necessary. The value of this approach is demonstrated by the adoption of the stack by projects other than the OSG Consortium itself. Technical leadership is demonstrated by the willingness of the external software development groups to accept requirements from OSG and contribute components to the software stack.

4.2. ***MyProxy***

MyProxy [67] is an open source software package that provides an authentication service, primarily for computational grids and other cyberinfrastructure utilizing public key infrastructures. It was developed at the National Center for Supercomputing Applications (NCSA) in 2000 and has now been sustained for a decade. It has found broad adoption across the NSF, DOE, and international communities, including EGEE [61], EU DataGrid [68], Earth System Grid [69], FusionGrid [70], LHC Computing Grid [71], NASA Information Power Grid [72], NCSA [73], NEESgrid [74], NERSC [75], Open Science Grid [59], and TeraGrid [76]. It has also had significant features contributed by developers from the European DataGrid, University of Virginia, and Lawrence Berkeley National Laboratory, as well as alternate (to the initial C-based implementation) language implementations in Python, Java, PHP, and Perl.

MyProxy is a clear success story for sustained software. The MyProxy team offers the following reasons for its success:

- *Clear, user-driven goals.* MyProxy was originally developed to solve a single well-defined problem: allow web portals to be used with Public Key Infrastructure (PKI)-based grids. Additions to MyProxy over the years have been selected based on well-defined and justified needs of the user community (and were often contributed by that community) and lack of disruption. This means carefully scrutinizing suggested changes and contributions and understanding their value and their disruptive impact before deciding if they should be incorporated.
- *Strong, consistent technical leadership.* MyProxy has benefited greatly by having a strong technical lead (Jim Basney) throughout its life, coordinating community contributions, ensuring the consistency of its architecture and processes, and minimizing any change-related disruption to the community.
- *Rapid prototyping and evolutionary design.* MyProxy was designed and developed quickly by a small team to fill an existing specific user need rather than by, for example, a standards committee working on a broad architectural problem. MyProxy has subsequently evolved based on user demand to solve a range of problems. While not excluding the possibility that other approaches may have worked, the approach of starting simply and evolving with careful scrutiny worked well for MyProxy.
- *Excellent software engineering practices.*
    - o *Software engineering, documentation, bug tracking, nightly builds, and regression tests.* These fundamentals are necessary, and not to be forgotten, but also not sufficient for sustaining software.
    - o *Maintaining backward compatibility and avoiding change for the sake of change.* There is often great social pressure to adopt new technology trends (e.g., web services, REST) or be seen as antiquated. The MyProxy team has avoided re-implementation and other changes in protocol or application programming interface (API) when that change didn't bring an obvious benefit to the user community. This contributed significantly to the fact that MyProxy has been remarkably backward

compatible over its 10-year lifetime.

  - *Maintaining compatibility with underlying libraries, protocols and standards.* Any software of reasonable complexity will rely on underlying libraries and standards. Balanced with the avoidance of jumping to new technologies for the sake of new technologies is maintaining compatibility with libraries and standards the software depends on. Maintaining releases and vulnerability fixes of those libraries and standards is often tedious work, but necessary to maintain trust and usefulness to the community.

- *Dissemination and community understanding.* Presentations, publications, and web presence make sure software is familiar to and understood by user communities. Much like commercial products, name recognition is valuable to making users feel comfortable with software.
- *Review and Accreditation.* Somewhat specific to security-oriented software, the fact that software has received external scrutiny and accreditation makes user communities (and in particular their operational security staff) more comfortable. MyProxy received a security assessment from the University of Wisconsin (via Bart Miller under NSF/NPACI funding) as well as having several of its deployments accredited by the International Grid Trust Federation [77].
- *Responding promptly and predictably to security vulnerabilities.* Related to the previous point, quick, effective responses to security issues are critical for maintaining user confidence. Predictable responses and coordination are important, particularly for large production deployments that typically want quick responses to vulnerabilities in conjunction with advance warning of public announcements of those vulnerabilities.
- *Encouraging collaborative software development.* Releasing the software as open source is only the first step to fostering collaboration and contributions. A simple, stable, documented software design with an open source implementation in a well-known language enables community members to modify the software to meet their needs. An open development process that is responsive to and appreciative of community contributions encourages community members to contribute back their changes. If the software is too complex or cutting-edge or frequently changing in fundamental ways, it is too difficult or time-consuming for external developers to participate. Stable, documented protocols (e.g., [78]) and APIs encourage developers to incorporate existing software in their work. Leveraging existing, well-defined standards (e.g., Transport Layer Security) and software (e.g., OpenSSL) aids in fostering collaboration.
- *Funding with emphasis on maintenance.* While MyProxy's open source status has resulted in a number of contributions from the community, the core maintenance of MyProxy has been made possible only by maintenance-focused funding. Maintaining software requires a great deal of familiarity with the software, requiring an investment in learning by personnel that makes maintaining staff critical. A typical problem in the research community is that maintenance funding is difficult to find. Many current funding mechanisms reward software creation, but not maintenance, resulting in proliferation of software that quickly quits being maintained and becomes hard to use. MyProxy funding has included a variety of National Science Foundation sources, including NCSA, the National Laboratory for Applied Network Research (NLANR), the NSF Middleware Initiative (NMI), the Strategic Technologies for Cyberinfrastructure Program (STCI), and the TeraGrid. Funding was also provided by NASA IPG.

Key lessons: A combination of excellent software engineering practices, strong technical leadership, and funding for sustainability and maintenance have enabled MyProxy to be highly valuable and widely used. MyProxy has benefitted from the need for and existence of an accreditation process certifying the functionality of the software, as well as from effective efforts to disseminate information about the software and its use.

4.3. *Sakai*

Sakai [23] is a learning and collaboration software environment distributed as open source and supported by a community source license. The Sakai Project was launched in early 2004 to create a collaboration and learning environment that could supersede then-current course management systems for education and incorporate collaboration tools for researchers. The project was initially funded by a $2.4M grant from the Andrew W. Mellon Foundation and $2.6M of in-kind staff investments from Indiana University, the University of Michigan, Massachusetts Institute of Technology, and Stanford University. The project developed several new models for software development that blended elements of directed development and open source models, though the software code itself was always distributed under a fully open license (first a BSD-style Educational Community License (ECL) 1.0, then the ECL 2.0 that is nearly identical to Apache save for lesser patent assertions).

By 2009, Sakai was in production use at many colleges and universities around the world. It had evolved a thriving developer community and a dozen Sakai Commercial Affiliates. The software scaled to production use in serving institutions of over 100,000 users and developed a pipeline of innovative tools to add on to the core release. Language ports include Japanese, Chinese, Spanish, and many others. New tools are exposed to the community as "Contrib," meaning they are available for use or improvement. Tools that mature, demonstrate production quality at one or more institutions, and garner community interest are promoted to "Provisional." Provisional tools can then be promoted to the Core Sakai release that goes through a community-based quality assurance process before release.

While the Sakai software remains freely available to anyone, the Sakai Project initiated the Sakai Partners Program in 2004 with 19 additional institutions that agreed to pay $10,000 per year ($5,000 for small schools) to help sustain the software and community.

Before the end of the Mellon grant in 2005, Sakai followed Apache and created the Sakai Foundation Inc. as a legally incorporated, not-for-profit, 501(c)(3) tax exempt organization to sustain the software. The foundation now has over 81 members from universities and commercial affiliates, and has distributed nine major releases of the software since August 2004. The Sakai Foundation operates by a members-elected board, an executive director, and a few staff who help coordinate member activity, quality assurance, and communications.

The Sakai model has been imitated and improved upon by the Kuali Foundation [24], which develops administrative system software for higher education. The practices of Sakai have demonstrated a path to core infrastructure for collaboration and learning environments for hundreds of colleges and universities around the world. It is a far less expensive option than commercial alternatives, and has proven to win in head-to-head comparisons [79].

The Sakai community attributes its success to several lessons that were learned in the rapid genesis of the project.

- Sakai's launch software was in production use at the University of Michigan in the first year of the project and at Indiana and other sites in year two. The first 24 months of the project yielded four major releases with rapid improvement. The momentum was essential to establish confidence for institutional adoption relative to more mature and well-financed commercial options.
- Deep engagement with commercial firms and treating them as valued and equal members of the Sakai community were critical in driving adoption, as commercial firms often led in marketing and essential communications about Sakai. They also contributed design and code, and helped redefine what had otherwise become a zero-sum vendor game.
- Sakai adopted the licensing practices of the Apache Foundation, with guidance from Brian Behlendorf, for Contributor Licensing Agreements to the Sakai Foundation and then unified licensing for redistributing the code.
- Sakai developed community processes that balanced inclusiveness, resources, timing, and quality in all core processes of design, development, quality assurance, documentation, and distribution.

Key lessons: Creating a lightweight coordination mechanism—the Sakai Foundation—was essential to provide a means to aggregate broad community resources and engagement to evolve the software. The aggregation of independent efforts across a core group of key contributing institutions created sufficient scale in effort and speed in implementation to enable early practical successes in use. Managing the Sakai brand to be a globally recognized brand of quality helped draw additional resources and adoption.

4.4. ***Findings Based on Examples from Successes in Sustainability***

From the examples above, from presentations and discussion at the workshop on *Cyberinfrastructure Software Sustainability and Reusability* (see Appendix 5), and from position papers submitted in advance, it is possible to draw two initial findings regarding cyberinfrastructure software sustainability:

Finding 1: A combination of focus on systematic collection and definition of user requirements and consistent application of good software engineering practices are important in enabling the sustained utility and sustainability of cyberinfrastructure software.

Finding 2: Because of the importance of reproducibility of scientific results and the challenges in maintaining software for small scientific communities over long periods of time, it is critical that cyberinfrastructure software be developed and released using an open source software license approved by the Open Source Initiative (http://www.opensource.org/), and that source code be managed and archived appropriately.

5. Cyberinfrastructure Software Is Infrastructure

The focus of this report, and the workshop on which it was based, is cyberinfrastructure software related to the mission of the National Science Foundation. The NSF's strategic plan, *National Science Foundation Investing in America's Future: Strategic Plan FY 2006–2011* [80], sets out four interrelated goals: discovery, learning, research infrastructure, and stewardship, depicted in Figure 1.

In particular, the NSF strategic plan describes "discovery" as "Foster[ing] research that will advance the frontiers of knowledge, emphasizing areas of greatest opportunity and potential benefit and establishing the nation as a global leader in fundamental and transformational science and engineering." Similarly the NSF strategic plan describes "research infrastructure" as "Build[ing] the nation's research capability through critical investments in advanced instrumentation, facilities, cyberinfrastructure and experimental tools."

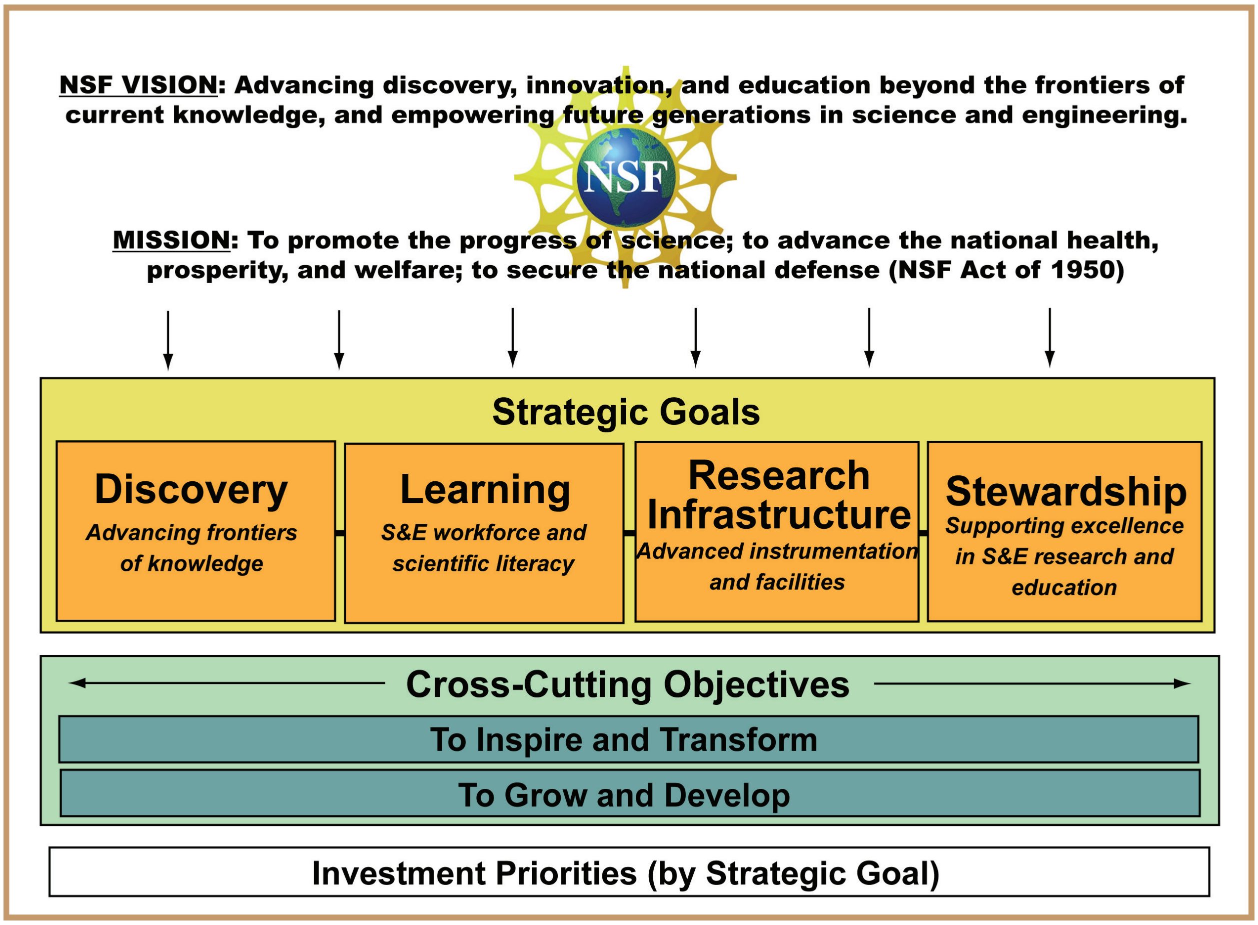

NSF VISION: Advancing discovery, innovation, and education beyond the frontiers of current knowledge, and empowering future generations in science and engineering.
NSF
MISSION: To promote the progress of science; to advance the national health, prosperity, and welfare; to secure the national defense (NSF Act of 1950)
Strategic Goals
Discovery
Advancing frontiers of knowledge
Learning
S&E workforce and scientific literacy
Research Infrastructure
Advanced instrumentation and facilities
Stewardship
Supporting excellence in S&E research and education
Cross-Cutting Objectives
To Inspire and Transform
To Grow and Develop
Investment Priorities (by Strategic Goal)

Figure 1. Graphical depiction of the NSF vision and mission, including for strategic goals [80].

As discussed by Neil P. Chue Hong in his talk at the workshop, one aspect of the creation of cyberinfrastructure software is the research and development of new software, creating new functionality or substantially improving on functionality already available. That is very much a 'discovery' process and rightfully funded as such through traditional peer review by the NSF or other funding agencies. It is to be expected that software research and development efforts funded by the NSF as a discovery process will routinely produce software that is of great general value, and particularly valuable as cyberinfrastructure software. Indeed, we see routinely that software created via NSF-funded research grants is of tremendous value to science and engineering research in the U.S. and globally. Open source software may deliver value to commercial entities in ways not initially anticipated, such as use of Condor for cycle scavenging in commercial settings [81]. However, the development of software as a research project is fundamentally different than the transition of such a research project through hardening to systematic longer-term support and maintenance of such software as cyberinfrastructure, and both efforts are important. As one rule of thumb holds, if developing software takes *x* amount of effort, hardening it for wide distribution will take approximately 3*x* effort and maintaining it will take 9*x* effort [82]. Studies cited in [83] suggest that the ongoing maintenance costs (as a proportion of total staff effort over a period) range between 49% and 75% depending on the industry. (A variety of very good references on software support are available at [84]).

When new software has been developed and its scientific value proved, and funding is requested for ongoing maintenance, improvement, and sustainability, then the basis for peer review and funding decisions for software should shift from the quality of the research to both the quality of the software itself and to the quality of the research the software can help enable. When this happens, software efforts clearly shift from being a discovery process to being a process of provisioning infrastructure.

The two findings presented below follow directly from the definitions of infrastructure, the NSF's depiction of its roles in four categories, and examination of solicitations for software development from the NSF in recent years. Both were strongly supported by discussion at the workshop on *Cyberinfrastructure Software Sustainability and Reusability:*

Finding 3: The NSF has generally been funding cyberinfrastructure software as if it were in the category of 'discovery'—funded through competitive peer-reviewed proposals based on the review criteria used for discovery proposals.

Finding 4: Cyberinfrastructure software is infrastructure—in particular, in terms of NSF strategic goal definition, cyberinfrastructure software is research infrastructure.

Similarly supported by discussion at the workshop, as well as points made earlier in this report, is the following finding:

Finding 5: There is some cyberinfrastructure software that is uniquely or primarily valuable to the NSF and NSF-funded researchers, in the same fashion as other unique research infrastructure funded and sustained by the NSF. A particular piece of cyberinfrastructure software generally has a longer lifespan than cyberinfrastructure hardware.

The first two recommendations in this workshop report derive directly from the first five findings and the exemplars of success described earlier:

> *Recommendation 1:* When funding software research and development, the NSF should put significant emphasis on the use of sound software engineering practices in evaluation of proposals and distribution of funding support.
>
> *Recommendation 2:* A condition of NSF support for creation and development of any cyberinfrastructure software should be the release of software under an open source license.

Release of software as open source is a requirement under some specific solicitations from the NSF and other funding agencies (such as the DOE's Scientific Discovery through Advanced Computing (SciDAC) [85]). As global collaborations become more and more critical to solving major science and engineering problems, open source facilitates such efforts. Global collaborations must, of necessity, seek common solutions. However, as discussed by Neil P. Chue Hong, there are a variety of forms of "free," and open source software tends to be of the form "free like a free puppy." Text cannot do justice to Dr. Hong's delivery of this analogy, but the key points are that open source software projects and puppies share the following characteristics: both are often interesting and engaging at first sight; there are long-term costs; need love and attention; may lose charm after growing up; occasional cleanups are required; and many are ultimately abandoned. The release of software as open source is helpful in enabling sustainability, but not sufficient to do so particularly because open source software released in isolation of other activities only ensures accessibility of the source code. Availability of software as open source does not ensure the capability to execute it, availability of the environment required for it to function, or possession of knowledge required to use it. This observation and the findings already described lead directly to a third recommendation—the most important specific recommendation in this entire report:

> *Recommendation 3:* Using the terminology of the National Science Foundation's four strategic goals outlined in its 2006–2011 strategic plan, the NSF should create funding mechanisms that support the ongoing sustainability and maintenance of cyberinfrastructure software as Research Infrastructure, employing mechanisms and evaluation criteria appropriate to Research Infrastructure rather than Discovery.

5.1. *Metrics of Use of Software as Infrastructure*

A key question that arises is how to measure the extent to which software is and functions as infrastructure. NSF has not in the past had a comprehensive approach to evaluation of software projects, and yet measurement of past accomplishments and understanding of future plans are essential in gauging the value of a particular piece of software as cyberinfrastructure. The workshop participants developed the following set of metrics to determine the extent to which cyberinfrastructure software functions as infrastructure:

- *Audience of users.*
    - Consider the current number of distinct users and expected rate of growth of users.
    - Consider the potential user community and expected growth of potential user communities.
    - Consider what percentage of that community (from outside the immediate project team) uses the code.
    - Consider both direct and indirect users (e.g., other code that depends on this code, and how widely that other code is used).
- *Group of creators.*
    - Determine the size of the community that is expected to contribute to and maintain the code.
    - Determine the breadth of the community that will continue to validate and then test the code.
- *Licensing terms.*
    - The software license selected should offer the greatest opportunity for collaboration within the research community over the anticipated life cycle of a particular project and beyond the life of a project.
- *Reusability.*
    - Determine what aspects of the software build upon existing resources that also comply with appropriate licenses as identified above.
    - Decide where a particular software package falls within the Reuse Readiness Levels [86] defined by the Goddard Space Flight Center Earth Science Data Systems Software Reuse Working Group.
    - Determine current or anticipated use of identifiable (reusable) components and dependencies.
- *Best practices in software engineering.*
    - Ongoing development and maintenance of the code should be

    managed within a formal software development plan, including milestones and metrics, and based on current best practices in software development.
  - Independent reviews and audits of software development should be conducted.

- *Functionality of software.*
  - A test suite should be distributed with the software to ensure that it is installed and functioning correctly.
  - In addition to simply running, software should function well.
  - Software should make efficient use of hardware resources.
  - In the case of parallel software, software should scale well.

- *Scientific outcomes.*
  - Determine what scientific outcomes the software has enabled.
  - Determine what publications the software has enabled, along with levels of citation and impact ratings for these publications.
  - Determine what major awards (e.g., Nobel Prizes) the software has enabled.

To consider cyberinfrastructure software as infrastructure implies that it is appropriate to measure it as such, using the criteria identified above and others as appropriate. These criteria may be set out in a tabular format as shown on page 25.

Finding 6: When cyberinfrastructure software is promoted as infrastructure (and/or there are requests to fund it as such), software should be measured and evaluated according to metrics relevant to it as infrastructure per se, such as the number of researchers who depend on use of such software. It is appropriate for the NSF to base decisions about whether or not to support the sustainability and maintenance of cyberinfrastructure software, and how much support to provide, on the basis of such metrics. Independent assessment of quality and impact should be strongly encouraged.

A proper and independent assessment of impact should be a formal and well-funded activity so that the assessments would stand up to standards of scientific review and security and process auditing.

Audience	**Current**	**Annual growth rate**	
Number of users?			
Potential user community?			
What percentage of the potential user community (outside of project team) uses the software?			
What other user communities use this software?			
Creators			
Size of community expected to contribute and maintain software			
Size of community that will test and validate code			
License terms			
What license terms are used?			
Reusability			
What is the current Reuse Readiness Level?			
What are current and anticipated uses of software components?			
	Module	**Depends on**	**License terms of software depended on**
What aspects of the software depend on other software?			
Best practices in software engineering			
Is there a formal software development plan?			
If so, what software development methodology is used?			
Are there independent reviews and audits of software development?			
If so, how often?			
Software functionality			
Is a test suite distributed with the software to verify proper installation and functionality?			
Describe the software's efficiency, including parallel scaling efficiency if appropriate.			
Scientific outcomes			
What publications have been enabled by this software?			
What major awards have been enabled by this software?			

6. Designing for Sustainability and Reusability

Much discussion at the workshop on *Cyberinfrastructure Software Sustainability and Reusability* focused on the realities of how CI software is developed today. Much of the coding for cyberinfrastructure software is done by students who have little formal training in software development, and in many cases the development of software is only a means to a research end in a domain science. Discussion about design for reusability and sustainability at the workshop included two major threads—one was about education for sustainability and reusability and the other was about characteristics of code development teams that promoted sustainability and reusability. One conclusion from a breakout session was that the cyberinfrastructure and computational science research community must be engaged and encouraged to produce "transition-ready" software. This idea, other discussions in the workshop, several presentations, and discussion in breakout sessions are consistent with and lead to the following finding:

Finding 7: Much software critical to the work of NSF-funded researchers is not developed or maintained in a way that corresponds to its importance to the science and engineering community. The science and engineering research community of the U.S. suffers from the fact that software is not developed and maintained in a way that is more sustainable. Behavior change on the part of developers and funding agencies is required to make cyberinfrastructure software more sustainable.

6.1. *Education for sustainability*

Considerable discussion throughout the workshop on *Cyberinfrastructure Software Sustainability and Reusability* centered on the question, "How can educators assist in the production of sustainable, reusable software through better education in computer science and software engineering?"

The Association for Computing Machinery (ACM) and IEEE Computer Society Computer Science Curriculum 2008 [87] emphasizes the increased importance of education in software engineering principles and techniques. Software engineering is the discipline concerned with efficiently building software systems that satisfy the requirements of users and customers, and is concerned with all phases of the software life cycle. The Curriculum emphasizes including the study of matters such as basic release management, basic source control principles, and best practice for developing software in teams. The Curriculum recommends a minimum of 31 core hours of instruction in software engineering for computer science and engineering majors, including coverage of software design; use and design of APIs, tools, and environments including source control and configuration management; validation and testing; and other core topics. In addition, software engineering is regarded as a subject applicable to the development of software in any computing applications domain. The report makes particular mention of the perceived benefits by industry of employing students who had contributed to open source software projects or who had other major software development experience. A Defense Advanced Research Projects Agency (DARPA)-funded project to study Software Engineering for Computational Science and Engineering [88] emphasizes the importance of software engineering in the context of computational science, and has resulted in a number of relevant publications [89-98].

More generally, the consensus of workshop attendees was that graduates should matriculate with the following concepts and skills (essential for computer science students, and highly desirable for students in domain sciences that make extensive use of cyberinfrastructure):

- Graduating students should understand that they will be developing code for a particular purpose and someone else will be using it. Similarly, they should have skill in reading and working with other people's code.
- Graduating students should understand that they will be developing code as part of a team and will need to interact directly with users. Students should be able to collaborate with team members and end users, and be proficient in bi-directional communication.
- Graduating students should have skill in gathering requirements and understanding scope (and managing against scope creep).
- Graduating students should be able to work in, and with, a modular framework for code development and when needed create modular code structures from scratch.
- Graduating students should have experience with version control, understanding code dependencies, and backtracking.
- Graduating students should have experience with testing.

Curriculum in software engineering should include best practices for software engineering. As mentioned earlier, use of best practices does not imply there is universal agreement on one best way to engineer software. Rather, there are several current contenders for best practices; use of one and awareness of the existence of others are extremely important.

Students who deal with software in computationally intensive sciences, but who are not programming students themselves, should be required to take a course in the basics of software management such as Software Carpentry [99, 100]. This course covers important and critical basic matters such as version control. Much as a large fraction of the increase in life expectancy in the past 200 years is based on very simple measures such as ensuring clean water and sanitation [101], uniform use of very basic and widely agreed-upon best practices in software engineering would enable considerable improvement in overall scientific practice related to software engineering. Discussions at and after the workshop led to two clear findings:

Finding 8: Efforts such as those documented in the 2008 Computing Curriculum to emphasize software engineering education and the Defense Advanced Research Projects Agency (DARPA)-funded project to study computational science and engineering are of considerable help in establishing basic principles of software engineering methodology and techniques into computationally oriented scientific disciplines.

Finding 9: The computer science, computational science, and computationally oriented scientific disciplines would benefit from the widespread adoption of one or a very few standard excellent textbooks or other learning materials in software engineering.

> *Recommendation 4:* The NSF should support joint efforts with organizations such as the *Association for Computing Machinery (ACM), the IEEE Computer Society, or Computing Research Association (CRA)*, incorporating the existing work done via DARPA support, to facilitate development of interdisciplinary courses and course materials on software engineering that are appropriate for computational science and for engineering students who are not computer scientists.

6.2. *Characteristics of software development teams and processes to create sustainability*

Through many discussions of the characteristics of programming teams and team behaviors, several consistent threads emerged. There was clear consensus that the following behaviors characterize leader and team behaviors and software development processes, engendering sustainability over the long run:

- The project should be planned with sustainability in mind. While this may involve many aspects of project management, particularly important aspects include the following:
    - There should be a project manager, and the project manager should have a vision for the project and for good architecture.

    - A formal plan for personnel resources, including succession plans for project leaders, should exist. Ideally, a software project should be so well documented and planned that it is portable—that is, it could be picked up from one group and moved to a different group for continued maintenance and/or development.
    - There should be a documented needs analysis and evidence of evaluation of existing codes on which to build.
    - There should be plans for software adoption as well as software development.

- Everyone on the project should start off their involvement knowing they are developing software intended to be sustainable over the long haul—which implies that it will be someday sustained by people other than those doing coding at any particular time.
- Some specific software development methodology should be used. While best practices in software architecture and development may change over time, workshop attendees concurred that the biggest quality difference is between software developed within some specific development methodology and coding done without reference to some development methodology. Software should be developed within some reasonable software development and testing practice, and any of the widely recognized software development methodologies in use today are better than development in absence of a software development methodology. Key characteristics of software methodologies include:
    - Code development must include use of a source control management system.
    - A project should employ an open, transparent architecture. It should be easy to comprehend and easy to assimilate by new leadership. This openness should cover all aspects.
    - Risk should be evaluated and managed; cyberinfrastructure software should generally be lower risk, which is counter to the current preference of many researchers.
    - Code should be clearly documented with up-to-date and accurate documentation of the software itself and dependencies on other software.
    - Software should be maintained and released with some sort of schedule and formal release procedure, including management of provenance of the code. Schedules for releases should be based on estimations of effort required, resources available, and statements of improvements needed. There should be a commitment to correct bugs in previous releases back to a certain version (at least one version back from current). There should similarly be a formal process and plan for ending the time during which the code is formally maintained.
    - Whenever possible, software should comply with relevant standards and be vendor agnostic.
- Software should be developed with clearly documented Application Program Interfaces (APIs).
- There should be a plan for promoting adoption of the software as well as the development of the software.
    - The project should include ongoing evaluation of the software. Formal surveys of users of the software—including questions about the software itself, software support by the group supporting it, and solicitation of suggestions for improvements and new features—should be included as part of such a survey.

There was considerable discussion of those behavior characteristics that do not promote sustainability. A general behavior offered as a stereotype—perhaps sometimes unfair but not always—is the academic researcher who wants to jump into coding to solve some research problem and publish as quickly as possible. There was clear agreement that designing for sustainability begins with the project leader responsible for a software project. A critical early decision point is whether a software developer (or developer team leader) is developing for her/himself alone, or intends to develop a tool with a broader user base. Of course, critical problems arise when someone intends to develop tools for themselves and then the software turns out to be of much broader use. Therefore, there was consensus on the following general recommendation on software development:

> *Recommendation 5*: All researchers developing software should do so using good software engineering practices, even if they intend to use the software only for themselves, but particularly if there is reason to believe that the software might evolve into a longer-lived infrastructure role.

This recommendation is perfectly consistent with text in the NSF "Cyberinfrastructure Vision for 21st Century Discovery," [2]: "NSF will promote the incorporation of sound software engineering approaches in existing widely-used research codes and in the development of new research codes." This recommendation goes against the natural tendency to just dive into a problem and start coding. We are, quite specifically, suggesting that extra effort be invested early in structuring any code. This is done in the belief that the net impact on individual researchers and on the science and engineering community will be positive. It is, going back to the software carpentry analogy, more than a bit like ensuring that beams in a house under construction be squared up before they are nailed in place. Whether such a house is used by one or many, the initial effort is in the long run worthwhile. So it is with good software structure and development practices.

7. Understanding Community Needs as a Tool in Sustainability

Software may acquire status as infrastructure in at least two ways: someone writes something so useful that it becomes widely used within a community or across multiple communities; or software is developed as part of a plan to provide infrastructure. Both middleware and application software may acquire status as CI software infrastructure. For example, middleware such as Condor or Globus are clearly middleware and CI software infrastructure. However, applications such as mathematics libraries, Sakai, Kuali, and LS-DYNA [102] may well be viewed as cyberinfrastructure software by segments of their user community. Throughout the workshop there was clear consensus on the value of and need for more focus on the gathering of needs and requirements as part of the software development and maturation process.

Indeed, while Recommendation 5 says in essence that the concept of good ideas without good software engineering is not sustainable, the opposite is also true—good software engineering practices with insufficient input from the researchers who will use the software is also not sustainable.

A focus on community needs runs through the success stories described in section 4. Other examples of current projects with a focus on community inputs include CaBIG [103], iPlant [104], and the Ocean Observatories Initiative [105]. These grants include the collection of community input as an activity funded explicitly within the grants that support cyberinfrastructure software. While it is too early to determine the long-term success of any of these projects, we believe the focus on NSF-funded elicitation of community needs constitutes an example that should be replicated whenever possible. Investment early on in defining community needs formally should result in software that better meets community needs overall, and better effectiveness in use of funds available to support the national science and engineering research community that depends upon cyberinfrastructure software.

> *Recommendation 6*: The NSF should establish and fund processes for collecting community requirements and planning long-term CI software roadmaps to support community research objectives. Such a process may be informed by, but is distinct from, an enunciation of grand challenge problems. When the NSF solicits proposals for software to be developed intentionally as cyberinfrastructure, such solicitations should call for funding for intensive and extensive stakeholder requirements determination as part of funded activities.

8. **Coming Changes in the Nature of Science and Scientific Reproducibility**

There was considerable discussion of the relationship between software sustainability and maintenance on the one hand and the nature of scientific research as regards software and reproducibility on the other hand. This discussion was informed particularly by a talk by Clifford Lynch, who contended that we are facing "fundamental methodological change in the practice of science" [106–108].

Lynch described at some length new steps taken by *Science* in 2006 following the submission and subsequent retraction of two fraudulent scientific papers. An external committee studied the process through which the fraudulent papers were vetted. *Science* followed a review process similar to that of other scientific journals, but the process was flawed. The committee recommended that *Science* strengthen its review procedures, particularly for papers of high public interest, those presenting unexpected results, and those that are potentially controversial. Some actions *Science* subsequently considered for high-risk papers include implementing higher standards for including primary data, requiring clear specification of author roles, and intense evaluation of digital images. Additionally, *Science* committed to developing criteria for a "risk assessment" template. This level of review may be successful for the rare case of scientific fraud. Much less onerous emotionally, and perhaps more difficult, are detecting and dealing with the impact of errors in software functionality created without ill intent and not noted by the software creators or users at the time a result based on that software is published.

We currently have no way (as Clifford Lynch put it) to "walk the cat backwards" and deal with the domino effect that takes place when software is documented to have some sort of problem or produce some sort of incorrect result. In a very different ethical context, we have processes for dealing with the domino effect of a paper being withdrawn as a result of falsification of data. Clifford Lynch laid out a set of recommendations indicating that we should develop processes for review of software, software artifacts, and data to prevent problems, and also to enable reanalysis of data when software is determined to have had some sort of fault. In particular, the following recommendations were made:

> *Recommendation 7*: The scientific community should promote scientific reproducibility. This can be done by requiring that: the provenance of software used in scientific research be carefully tracked and that versions used in particular experiments be documented in scientific publications; software, data, and software and data artifacts used in analyses should be available for review along with the text of a scientific publication as part of the peer-review process prior to publication; and data and software used in the development of a scientific publication should be escrowed or archived where they can be examined and re-verified when needed.

It is particularly important for scientific reproducibility to manage the preservation and provenance of data and software together. It should be noted that provenance collection is still an active research area though the provenance community is making rapid strides for instance through agreement on an Open Provenance Model [109]. In this regard, a stronger form of the above recommendation can be made specifically for NSF-funded research:

> *Recommendation 8*: The NSF should require that data and software used in the development of a scientific publication based on NSF-funded

research be escrowed or archived where it can be examined and re-verified as appropriate, in order to enable robust verification and reproducibility of scientific findings in the face of the cyberinfrastructure-dependent research environment of the 21st century.

Here we are making a recommendation that applies to cyberinfrastructure software, but is also applicable beyond cyberinfrastructure software to any science application software. Because of the complexity of cyberinfrastructure software, it may be that reproducibility of the underlying cyberinfrastructure software is the more difficult part of the problem to solve. Examples of current efforts to enable the archiving and reuse of cyberinfrastructure-based research include nanoHUB [110], Programmableweb.com [111], and myexperiment.org [112]. Indeed, the NSF-funded DataNet program may develop some of the technology needed to begin creating something akin to a national system of "libraries of scientific data, software, and data artifacts."

Another issue related to reproducibility is the fact that software generally outlives hardware; thus, a particular piece of software may well be available long after the end of the life span of the particular system for which it was written. Long-term preservation of software can be achieved without the original hardware by either emulating the original hardware or migrating the software to a contemporary hardware/software platform. The latter strategy requires a high ongoing expense to port programs and libraries as well as access to the required source code. In contrast, hardware emulation is a proven technology with broad applicability.

Hardware emulation simulates a hardware platform, including the processor and input/output (I/O) devices, in such a way that an operating system and application software can perform as if executing on the hardware itself. Instruction-set simulation does not typically impose a significant performance burden. Techniques such as on-the-fly translation of instructions have been successfully implemented with excellent performance [113] and form the basis of at least one x86 "clone" [114]. In fact, prior to recent additions to the x86 architecture to support virtualization, some instruction-set translation was required to support x86 guests on x86 hosts [115].

Virtualization provides an additional layer of abstraction. For example, the Amazon Elastic Compute Cloud (EC2) [116] utilizes virtualization to deliver compute services, and many IT organizations utilize virtualization to deliver core services such as email and web servers. Virtualization software is available both in open-source and commercial forms. Examples of commercial virtualization platforms include VMWare [117], Xen [118], and Parallels [119], while open-source tools include Qemu [120], Bochs [121], and Plex86 [122]. In practice, virtualization is a special case of emulation in which the processor instruction sets of the emulation platform (host) and the emulated machine (guest) are sufficiently well matched that little instruction-set simulation is required. The fundamental challenge for implementing virtualization is in emulating the I/O devices such as graphics and networking hardware. A good summary of some of these issues is available in [123].

While emulation is a viable and widely deployed technology that can provide reproducibility of results (other than performance analyses per se), there are two fundamental limitations—technical and social—that must be addressed in the context of software sustainability. First, as noted above, the "difficult" issue with emulation is preserving the I/O mechanisms including networking. All existing virtualizations provide a core set of relatively generic video, storage, and networking devices. Software that depends upon specific and unusual I/O devices, for example, programs that directly access high-performance graphics processing units, are not supported by existing emulation platforms and are unlikely to be in the future. This is because such devices achieve their performance through special-purpose hardware that cannot be simulated with reasonable performance. However, such hardware dependencies also make this software difficult to port to new platforms. Also, the ever-growing use of highly parallel machines raises significant reproducibility problems that become much more severe in emulation or virtualization situations.

As regards the social limitation, many programs implicitly rely upon contemporary user interfaces for access. For example, it is unlikely that many users today remain facile in their interaction with MS-DOS. A possible solution for sustaining legacy software is to utilize scripts in an emulation to simplify this interaction for contemporary users [124].

This discussion leads directly to two findings and one recommendation:

Finding 10: Hardware emulation provides an excellent mechanism for enabling reuse of software after the hardware on which it was originally run no longer exists. Unresolved issues remain in terms of emulating special purpose processors, I/O generally, and network environments in particular.

The chances of future reproducibility of software are increased by developing software to use well-defined and portable APIs rather than accessing platform specific hardware directly. This will not always be possible, for example in cases of extremely performance-intensive applications or novel architectures.

Finding 11: User interfaces evolve rapidly over time and older interfaces are relatively quickly forgotten.

> *Recommendation 9*: Sustainable software should be built with user interfaces that do not functionally depend upon the user having extensive knowledge of the software's operating environment.

A final point on software: it is clear that data have a life cycle. Software does as well. Attention tends to focus on the early parts of the life cycle, when the software is still actively being used, maintained, and enhanced. It is equally important to pay attention to the latter part of the life cycle, when use is tapering off and eventually ends, and to ensure that at this point software is archived in appropriate ways and for appropriate lengths of time (though, as with data, not necessarily in perpetuity).

9. NSF Funding Behaviors and Sustainability

Throughout the workshop on *Cyberinfrastructure Software Sustainability and Reusability* there were discussions of what it is that is valuable to sustain. There are at least three types of entities that are valuable to sustain over time as part of software sustainability and reusability: functionality, software, and people.

For example, authentication and file transfer are treated largely as capabilities. Researchers on average care more about getting the tasks done than about how they are done. Discussion in plenary and breakout sessions supported the following finding and recommendation:

Finding 12: In some cases, what researchers and practitioners care about having sustained is not a particular piece of software but rather a required capability.

> *Recommendation 10*: The NSF should be prepared to make decisions to fund a succession of software, over time, that provide key required capabilities and in so doing focus on a limited number of robust codes maintaining a particular functionality at any given time.

It was suggested at the workshop that the NSF should be prepared to fund fewer experimental discovery projects in computer science software development and make fewer but larger awards, driving computer scientists more toward creating one solution to solve community needs by writing software for the community of users more and writing software as a matter of personal experimentation less. While certain challenges inherent in this approach were noted, there was also a good bit of support for this general approach. This is consistent with one of the key conclusions of Neil P. Chue Hong's talk—that increasing utilization of a particular piece of software is a key to sustainability.

Drawing specifically from the talks by Neil P. Chue Hong and Brad Wheeler, it is possible to identify an order in the development of software—not one that is universally followed, but one that characterizes the development of software:

- Funding as a research project
- Heroic research
- Everyday research
- Production use
- Sustainability (through a variety of possible means)

A clear undercurrent in discussions throughout the workshop was the belief that commercialization is not a viable path for sustainability of many sorts of cyberinfrastructure software critical to U.S. national competitiveness and NSF-funded science and engineering research. The general success rate of past efforts to commercialize such software—in some cases software well engineered and heavily used in scientific research—supports this. Some software essential to the NSF mission must be sustained by paths other than commercialization.

Other entities beyond software functionality must be sustained over the long run to support an innovative and effective national cyberinfrastructure. It is a common observation that graduate students and postdoctoral fellows are often used to write code as a seemingly cheaper alternative to professional programmers—cheaper perhaps in the short run and perhaps not in the long run, due to the costs over the long haul of using code not well engineered and not designed for reusability. In fact, it is very difficult to measure the real cost of software development because much cost is hidden in the use of students and postdoctoral fellows to write code. One of the entities that the U.S. science and engineering community should sustain and maintain is expertise in the science and engineering software development community. Craig Stewart testified at a hearing of the House Science and Technology Committee in July 2008 [125], on behalf of the Coalition for Academic Scientific Computation (CASC), and spoke specifically on this point. Stewart said:

> "Without strong, continued, and consistent investment in networking and information technology (NIT), the U.S. will not have the administrative and technical leadership to support consistent and directed change. Government investment in NIT will be of greatest value if there is consistency in levels of investment over time. The men and women who execute the national NIT agenda represent a tremendous store of experience, skill, and knowledge. The uniform experience of CASC members is that when there are strong variations in funding in specific areas of NIT over time, lean times for particular areas of research in NIT cause skilled professionals to leave public sector NIT research. This means that years of investment by the government in developing a knowledge and experience base in individuals who desire to pursue a career in the public service sector are lost to the public sector, not to return even when funding for particular areas is subsequently restored. U.S. global competitiveness, innovation, and homeland security are thus best served by consistent and strong investment in basic NIT research; advanced NIT facilities to support advanced research and development in science, engineering, and technology; and research in developing and delivering the next generation of such advanced NIT facilities."

There was also clear and unambiguous emphasis throughout discussion at the workshop on the value of support for use of software, by knowledgeable experts, as a key part of sustaining software. There are three intertwined concerns: supporting the sustainability of software itself; supporting the use of said software; and supporting professional code developers and support personnel. The report "Understanding Infrastructure: Dynamics, Tensions, and Design" [126] was the final report of an NSF-funded workshop on "History & Theory of Infrastructure: Lessons for New Scientific Cyberinfrastructures." This report states, "robust cyberinfrastructure will develop only when social, organizational, and cultural issues are resolved in tandem with the creation of technology-based services. Sustained and proactive attention to these concerns will be critical to long-term success."

Direct support from NSF for cyberinfrastructure software seems well within scope of NSF activities described in the "Cyberinfrastructure Vision for 21st Century Discovery" [2], which includes within its mission for cyberinfrastructure the following activities:

- "Provide the science and engineering communities with access to world-class CI tools and services, including those focused on: high performance computing; data, data analysis and visualization; networked resources and virtual organizations; and learning and workforce development.
- "Provide a sustainable CI that is secure, efficient, reliable, accessible, usable, and interoperable, and that evolves as an essential national infrastructure for conducting science and engineering research and education.
- "Create a stable but extensible CI environment that enables the research and education communities to contribute to the agency's statutory mission."

Such direct support by the NSF would also be within the letter and spirit of recommendations made in the 2000 President's Information Technology Advisory Committee report on open source software and high end computing [127].

While the issues of sustaining software, sustaining personnel and organizational expertise, and support for software are necessarily conflated to a significant extent, it is possible to make one recommendation that addresses sustainability of functionality, software, people, and usability:

> *Recommendation 11:* The NSF should create a funding program to fund CI software development, hardening, support, and sustainability. Such a program might be based on long-term funding for "software centers" modeled after the existing Science and Technology Center program.

Using the example of the Commissioned Software Programme of the Open Middleware Infrastructure Institute (OMII) discussed by Neil P. Chue Hong, such centers could manage effort and priorities in part on the basis of "effort tokens." A token implies the application of a certain amount of effort on the part of the software development program, with funding built into the budget to fulfill such promises. Effort tokens are then awarded by the funding agency to users of the software. This approach drives a user-centric focus to the maintenance, sustainability, and further development of software. This approach would offer improved predictability of funding for personnel and expert support for users, in addition to sustainability of cyberinfrastructure software itself.

At the same time, support from the NSF cannot be the only vehicle for sustaining software. Cyberinfrastructure development projects can likely benefit from the foundation community model used widely as regards operating system software development, the Sakai and R foundations, and the promising start of the HUBzero consortium. Foundations create entities, separate from individual projects, that can exhibit considerable longevity. There is the initial cost of creating a charter, organizing documents, and establishment of a legal entity. At the same time, such a foundation established as a legal entity may have a higher likelihood of being sustainable as a result of the existence of and ability to interact with a foundation rather than ad hoc interactions among a population of loosely related projects. This leads to the following recommendation:

> *Recommendation 12:* The U.S. research community should, when feasible, pursue the model of collaborating within the framework of a not-for-profit foundation as a way to maintain and sustain cyberinfrastructure software development and support.

Early examples of a foundation approach, rooted in the common interests of collaborating institutions of higher education, have been promising. New foundations built around specific themes may follow the promising examples of the Apache, R, Sakai, and HUBzero organizations. Such foundations would not necessarily have to run their own code repositories; new foundations might conceivably use the Apache foundation software repository for that purpose.

More ambitiously, it might be possible that universities could band together to create a more general programming foundation. Many speakers and participants at the workshop made note of the fact that the scientists who develop software as part of their research activities may not want to take on the role of ongoing software maintainers and supporters. A more general university collaboration could create something like the 'Educore' service proposed by Fuchs [128]. Specifically, Fuchs called for a collaborative university-funded organization that would "…coordinate the development and maintenance of open source for the benefit of higher education." Such an organization would provide scientists who create innovative and useful software an organization that could take on the ongoing software sustainability and support tasks.

Recommendations 12 and 13, and Fuch's suggestion for an 'Educore' program, are all consistent with text included in the NSF "Cyberinfrastructure Vision for 21st Century Discovery," [2] specifically:

"The software provider community will be a source for: applied research and development of supporting technologies; harvesting promising supporting software technologies from the research communities; performing scalability/reliability tests to explore software viability; developing, hardening and maintaining software where necessary; and facilitating the transition of commercially viable software into the private sector. It is anticipated that this community will also support general software engineering consulting services for science and engineering applications, and will provide software engineering consulting support to individual researchers and research and education teams as necessary."

Last, and definitely not least, it is important to recognize that software sustainability is in some sense still a national and international social, scientific, and financial experiment. As discussed at the workshop, and particularly as recommended by Clifford Lynch, a final recommendation arises for the NSF as regards funding:

> *Recommendation 13*: The NSF should fund empirical studies of software sustainability efforts, so that the NSF, other funding agencies, and the science and engineering community generally can learn from and build upon real-world experience in developing and supporting cyberinfrastructure sustainably.

Such studies should specifically address return on investment, and may well need to take place over the course of several years so as to have sufficient perspective and data to allow the correct, long-term conclusions to be drawn.

10. **Final Notes**

There was considerable discussion throughout the workshop of position papers submitted in advance, presentations, and ideas brought up in discussion. Each presenter's slides are included in Appendix 5.

All of the position papers submitted in preparation for the workshop on *Cyberinfrastructure Software Sustainability and Reusability* are included as part of Appendix 1. The recommendations contained in each position paper are collated as Appendix 2. Ultimately it was not possible to have a comprehensive discussion of each position paper and the recommendations each contained. However, an online poll was taken during the course of the workshop to ascertain which recommendations workshop participants viewed as most important.

The 10 most popular recommendations from the position papers were as follows:

1. A free flow of information between the global open source community, industry, and academia should be created. Extensible and interoperable CI development should be encouraged.
2. The NSF should establish evaluation criteria and funding mechanisms that support software development, release, and life-cycle improvement. This is particularly critical for relatively lower-use software that is essential to the nation's eScience objectives but which may not initially have a broad user base or immediate commercial potential. Funding should be provided to support software development technologies including repositories user mailing lists, bug-tracking, and testing.
3. Funding agencies should award grants to software development after the research phase is done, to sponsor long-term sustainable development.
4. The open source community software should be supported through investments of time from developers and monies from grants.
5. The NSF should permit funding for software incubation, development, and support to be included in future CI proposals, in particular those proposals that are directed at the development of community-oriented CI products such as, but not limited to, innovative parallel libraries, domain-specific grid "stacks," storage management, collaboration tools, visualization (including remote visualization), and portal components.
6. Advance discussions on software interoperability and dissemination should be aggressively encouraged.
7. The same level of detailed oversight should be used for software licensing/development awards as is used for hardware procurement and installation.
8. The NSF should invest heavily in infrastructure that facilitates collaboration for researchers and sharing of data tools and results.
9. Agencies should strive to create user and developer communities around software, as they are just as important as the actual software development project.
10. Software infrastructure projects in particular should use the open source model.

All of these recommendations are addressed either explicitly or implicitly in recommendations already stated in this workshop report except two—the number 1 and number 6 most supported recommendations. Thus, a final recommendation in this workshop report embodies the sense of both of these recommendations, and discussion related to them during the course of the workshop:

> *Recommendation 14:* The NSF should encourage and support the flow of information between the global open-source community, industry, and academia, focused particularly on encouraging extensible and interoperable CI development, through support for development and maintenance of software standards when appropriate.

Finally, at the workshop and in subsequent discussion of draft reports via email there were several important themes and in some cases interesting discussions which garnered significant support but not general consensus. Among these points were the following:

- Workshop participants expressed interest in additional reports and/or workshops on the particular issues of domain science codes and their sustainability.
- Workshop participants expressed interest in additional reports and/or workshops on the particular role of international collaborations as a tool toward software sustainability, and other approaches to sustainability of interest in communities beyond the NSF and the U.S. science and engineering community.
- There was considerable discussion of much stronger recommendations about computer science and/or software engineering requirements for students in domain sciences. Workshop participants were not able to arrive at general consensus on a stronger wording, however.

11. **Acknowledgements**

Bradley C. Wheeler, Beth Plale, Geoffrey Brown, Craig Stewart, and Therese Miller (the PI and co-PIs of the grant that funded this workshop) wish to acknowledge the support of the National Science Foundation. We would like also to particularly thank all of the individuals who gave presentations:

Brad Wheeler
Vice President for Information Technology, CIO and Dean
Professor, Kelley School of Business
601 Kirkwood Avenue, FH 116
Bloomington, IN 47405–1223
http://www.indiana.edu/~ovpit/
bwheeler@indiana.edu

Jennifer Schopf
Woods Hole Oceanographic Institution
266 Woods Hole Road
Woods Hole, MA 02543
jschopf@whoi.edu

Dennis Gannon
Director of Applications, Cloud Computing Futures Center
Bldg 99/2438
One Microsoft Way
Redmond, WA 98052
http://research.microsoft.com/en-us/groups/ccf/default.aspx
Dennis.Gannon@microsoft.com

Neil P. Chue Hong
Director, OMII-UK
Rm. 2409, JCMB, Mayfield Rd. | E:
Edinburgh, EH9 3JZ, UK
http://www.omii.ac.uk
N.ChueHong@omii.ac.uk

Clifford Lynch
Executive Director
Coalition for Networked Information
21 Dupont Circle, Suite 800
Washington, D.C. 20036
http://www.cni.org
cliff@cni.org

We would also like to thank the individuals who coordinated breakout sessions:

John Campbell, Geoffrey C. Brown, Andrew Lumsdaine, Brad Wheeler, Tony Drummond, Beth Plale, Sudhakar Pamidighantam, and Shelton Waggener.

We would like to thank particularly Peg Lindenlaub, Jennifer Browning, David Donaldson, Michael Grobe, Michael Jasiak, Richard Knepper, Dale Lantrip, Matt Lewallen, Malinda Lingwall, Nancy Long, Don Lorentz, Sidd Maini, Joe Rinkovsky, Josh Rosen, Ray Sheppard, and Matt Standish especially for their extraordinary efforts supporting this workshop and the creation of this report.

This material is based upon work supported by the National Science Foundation under Grant No. OCI-0829462. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

This material and the creation of this report are funded in part by a generous grant from the Lilly Endowment, Inc. to Indiana University to create the Pervasive Technology Institute. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Lilly Endowment, Inc.

12. **References**

1. Blatecky, A. and D. Messerschmitt. Planning for Cyberinfrastructure Software. February 2005. Available from: http://www.nsf.gov/od/oci/ci_workshop/workshopreport_final_rev2a.pdf [cited 19 Jan 2010]

2. National Science Foundation Cyberinfrastructure Council. Cyberinfrastructure Vision for 21st Century Discovery. March 2007. Available from: http://www.nsf.gov/pubs/2007/nsf0728/index.jsp [cited 19 Jan 2010]

3. Colwell, R. "Information Technology: Ariadne's Thread through the Research and Education Labyrinth*." EDUCAUSE Review*, May/June 2000. **35**(3): p. 14–18. Available from: http://www.educause.edu/ir/library/pdf/erm0030.pdf [cited 19 Jan 2010]

4. Cornell University Law School. U.S. Code: Title 35,200. Policy and objective. 8 Jan 2008. Available from: http://www4.law.cornell.edu/uscode/35/200.html [cited 19 Jan 2010]

5. Wessner, C. W., ed. *An Assessment of the SBIR Program at the National Science Foundation*. 2007, National Academy of Sciences: Washington, DC. Available from: http://www.nap.edu/catalog.php?record_id=11929 [cited 19 Jan 2010]

6. Evans, J. and G. P. Schneider. *New Perspectives on the Internet, Brief*. 7th ed. 2008, Cambridge, MA: Course Technology.

7. Rafinejad, D. *Innovation, Product Development and Commercialization: Case Studies and Key Practices for Market Leadership*. 2007, Fort Lauderdale, FL: J. Ross Publishing.

8. SPSS. Home page. Available from: http://www.spss.com/ [cited 3 Feb 2010]

9. MapleSoft. Math Software for Engineers, Educators & Students. Available from: http://www.maplesoft.com/ [cited 19 Jan 2010]

10. Courant, P. N. and R. J. Griffiths. Software and Collaboration in Higher Education: A Study of Open Source Software. [PDF] 26 Jul 2006. Available from: http://www.ithaka.org/ithaka-s-r/strategy/oss/OOSS_Report_FINAL.pdf [cited 19 Jan 2010]

11. Weber, S. *The Success of Open Source*. 2004, Cambridge, MA: Harvard University Press.

12. Karels, M. "Commercializing Open Source Software*." ACM Queue*, 2005. **1**(5): p. 46–55.

13. Feller, J., B. Fitzgerald, W. Scacchi, and A. Sillitti, eds. *Open Source Development, Adoption and Innovation: IFIP Working Group 2.13 on Open Source Software*. 2007, IFIP International Federation for Information Processing: Limerick, Ireland.

14. OSS Watch: open source advisory service. Available from: http://www.oss-watch.ac.uk/ [cited 19 Jan 2010]

15. Wheeler, B. C. "Open Source 2010: Reflections on 2007*." EDUCAUSE Review* 2007. **42**(1): p. 48–67. Available from: http://connect.educause.edu/Library/EDUCAUSE+Review/OpenSource2010Reflections/40682 [cited 19 Jan 2010]

16. Perens, B. "The Emerging Economic Paradigm of Open Source*." First Monday, Special Issue #2: Open Source*. Available

from: http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1470/1385 [cited 19 Jan 2010]

17. Gandel, P. and B. C. Wheeler. "Of Birkenstocks and Wingtips: Open Source Licenses*." EDUCAUSE Review*, 2005. **40**(1): p. 10–11. Available from: http://connect.educause.edu/Library/EDUCAUSE+Review/OfBirkenstocksandWingtips/40529 [cited 19 Jan 2010]

18. Mann, R. "Commercializing Open-Source Software: Do Property Rights Still Matter?*" 20 Harv. J.L. & Tech. 1*, 2006.

19. Valimaki, M. *The Rise of Open Source Licensing: A Challenge to the Use of Intellectual Property in the Software Industry*. 2005, Helsinki: Turre Publishing.

20. SourceForge. Home page. Available from: http://sourceforge.net/ [cited 19 Jan 2010]

21. Apache Software Foundation. Home page. Available from: http://www.apache.org/ [cited 19 Jan 2010]

22. Wheeler, B. C. "Open Source 2007: How Did This Happen?*" EDUCAUSE Review*, 2007. **39**(4): p. 12–27. Available from: http://connect.educause.edu/Library/EDUCAUSE+Review/OpenSource2007HowDidThisH/40482 [cited 19 Jan 2010]

23. Sakai. Home page. Available from: http://sakaiproject.org/ [cited 19 Jan 2010]

24. Kuali Foundation. Home page. Available from: http://www.kuali.org/ [cited 19 Jan 2010]

25. The R Foundation. Home page. Available from: http://www.r-project.org/foundation/main.html [cited 19 Jan 2010]

26. HUBzero Platform for Scientific Collaboration. Home page. Available from: https://hubzero.org/home [cited 19 Jan 2010]

27. Lundstrom, M. and G. Klimeck. "The NCN: Science, Simulation, and Cyber Services." In: *2006 IEEE Conference on Emerging Technologies—Nanoelectronics.* p. 496–500. 2006.

28. Klimeck, G., M. McLennan, S. P. Brophy, G. B. Adams III, and M. Lundstrom. "nanoHUB.org: Advancing Education and Research in Nanotechnology*." Computing in Science and Engineering*, September/October 2008. **10**(5): p. 17–23.

29. Coalition for Networked Information. Home page. Available from: http://www.cni.org/ [cited 19 Jan 2010]

30. Council on Library and Information Resources. Home page. Available from: http://www.clir.org/ [cited 19 Jan 2010]

31. Digital Curation Centre. Home page. Available from: http://www.dcc.ac.uk/ [cited 3 Feb 2010]

32. EDUCAUSE Campus Cyberinfrastructure Working Group and Coalition for Academic Scientific Computation. *Developing a Coherent Cyberinfrastructure from Local Campus to National Facilities: Challenges and Strategies*. February 2009. Available from: http://www.educause.edu/Resources/DevelopingaCoherentCyberinfras/169441 [cited 19 Jan 2010]

33. Wikipedia. Cyberinfrastructure. Available from: http://en.wikipedia.org/wiki/Cyberinfrastructure [cited 19 Jan 2010]

34. *"*infrastructure*." In: Merriam-Webster Online Dictionary*. Available from: http://www.merriam-webster.com/dictionary/infrastructure [cited 26 Jan 2010]

35. Clarke, R. and J. Hunker. Press Briefing by Richard Clarke, National Coordinator for Security, Infrastructure Protection, and Counter-Terrorism; and Jeffrey Hunker, Director of the Critical Infrastructure Assurance Office. 1998. Available from: http://www.fas.org/irp/news/1998/05/980522-wh3.htm [cited 28 Jan 2010]

36. Atkins, D. E., K. K. Droegemeier, S. I. Feldman, H. Garcia-Molina, M. L. Klein, D. G. Messerschmitt, P. Messina, J. P. Ostriker, and M. H. Wright. *Revolutionizing Science and Engineering Through*

Cyberinfrastructure: Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure. 2003. Available from: http://www.nsf.gov/od/oci/reports/toc.jsp [cited 28 Jan 2010]

37. epcc. Glossary. Available from: http://www.epcc.ed.ac.uk/other-information/glossary/ [cited 26 Jan 2010]

38. Condor Project. Condor High Throughput Computing. Available from: http://www.cs.wisc.edu/condor/ [cited 19 Jan 2010]

39. Globus Alliance, T. Home page. Available from: http://globus.org/ [cited 26 Jan 2010]

40. Virginia Center for Grid Research. Home page. Available from: http://www.cs.virginia.edu/~vcgr/wiki/index.php/Main_Page [cited 26 Jan 2010]

41. MPICH. Home page. Available from: http://www.mcs.anl.gov/research/projects/mpi/mpich1/ [cited 29 Jan 2010]

42. Open MPI Project. Open MPI: Open Source High Performance Computing. Available from: http://www.open-mpi.org/ [cited 29 Jan 2010]

43. Cactus Code. Home page. Available from: http://cactuscode.org/ [cited 26 Jan 2010]

44. Pegasus. Home page. Available from: http://pegasus.isi.edu/ [cited 29 Jan 2010]

45. Storage Resource Broker. Home page. Available from: http://www.sdsc.edu/srb/index.php/Main_Page [cited 28 Jan 2010]

46. iRODS. IRODS: Data Grids, Digital Libraries, Persistent Archives, and Real-time Data Systems. Available from: https://www.irods.org/ [cited 28 Jan 2010]

47. freshmeat. GotoBLAS. Available from: http://freshmeat.net/projects/gotoblas/ [cited 26 Jan 2010]

48. FFTW. Home page. Available from: http://www.fftw.org/ [cited 26 Jan 2010]

49. DSpace. Home page. Available from: http://www.dspace.org/ [cited 3 Feb 2010]

50. FedoraCommons. Fedora Repository. Available from: http://www.fedora-commons.org/ [cited 3 Feb 2010]

51. EPrints. Open Access and Institutional Repositories with EPrints. Available from: http://www.eprints.org/ [cited 3 Feb 2010]

52. Microsoft Corp. Microsoft Office Word. Available from: http://office.microsoft.com/en-us/word/default.aspx [cited 29 Jan 2010]

53. Corel Corp. WordPerfect Office X4. Available from: http://www.corel.com/servlet/Satellite/us/en/Product/1207676528492 [cited 29 Jan 2010]

54. "sustain*." In: Merriam-Webster Online Dictionary*. Available from: http://www.merriam-webster.com/dictionary/sustain [cited 19 Jan 2010]

55. "sustainable*." In: Merriam-Webster Online Dictionary*. Available from: http://www.merriam-webster.com/dictionary/sustainable [cited 19 Jan 2010]

56. JISC. Models of Sustainability Workshop. 2007. Available from: http://www.jisc.org.uk/whatwedo/programmes/einfrastructure/modelsofsustainability.aspx [cited 22 Jan 2010]

57. Knowles, A., N. C. Hong, C. Goble, J. Austin, A. Sudlow, S. Lloyd, Z. Lock, T. Linde, and J. Nicholson. *UK e-Science Core Programme: Business Models for Sustainability*. Available from: http://www.jisc.ac.uk/media/documents/programmes/einfrastructure/day2_breakoutbusinessmodels.pdf [cited 22 Jan 2010]

58. HECToR: UK National Supercomputing Service. Home page. Available from: http://www.hector.ac.uk/ [cited 19 Jan 2010]

59. Open Science Grid. Home page. Available from: http://www.opensciencegrid.org/ [cited 19 Jan 2010]

60. Virtual Data Toolkit. GUMS. Available from: http://vdt.cs.wisc.edu/components/gums.html [cited 19 Jan 2010]

61. EGEE–Enabling Grids for EscienceE. gLite.

Available from: http://glite.web.cern.ch/glite/ [cited 19 Jan 2010]

62. Virtual Data Toolkit. Home page. Available from: http://vdt.cs.wisc.edu/ [cited 19 Jan 2010]

63. Open Science Grid. Software. Available from: http://www.opensciencegrid.org/OSG_at_work/Software [cited 19 Jan 2010]

64. Pacman Headquarters. Home page. Available from: http://atlas.bu.edu/~youssef/pacman/ [cited 19 Jan 2010]

65. NMI Build and Test Lab. Home page. Available from: https://nmi.cs.wisc.edu/node/160 [cited 19 Jan 2010]

66. NMI Build and Test Lab. What is Metronome? Available from: https://nmi.cs.wisc.edu/node/90 [cited 19 Jan 2010]

67. Basney, J., M. Humphrey, and V. Welch. "The MyProxy Online Credential Repository*." Software: Practice and Experience*, July 2005. **35**(9): p. 801–816. Available from: http://www3.interscience.wiley.com/cgi-bin/fulltext/110541011/PDFSTART [cited 19 Jan 2010]

68. The DataGrid Project. Home page. Available from: http://eu-datagrid.web.cern.ch/eu-datagrid/ [cited 13 May 2010]

69. Earth System Grid. Home page. Available from: http://www.earthsystemgrid.org/home.htm [cited 13 May 2010]

70. FusionGRID. Home page. Available from: http://www.fusiongrid.org/ [cited 13 May 2010]

71. Worldwide LHC Computing Grid. Home page. Available from: http://lcg.web.cern.ch/lcg/ [cited 13 May 2010]

72. GLORIAD. NASA Information Power Grid (IPG) Infrastructure. Available from: http://www.gloriad.org/gloriad/projects/project000053.html [cited 13 May 2010]

73. NCSA. Home page. Available from: http://www.ncsa.illinois.edu/ [cited 13 May 2010]

74. Network for Earthquake Engineering Simulation. NEES Cyberinfrastructure. Available from: https://www.nees.org/it/ [cited 13 May 2010]

75. National Energy Research Scientific Computing Center (NERSC). Home page. Available from: http://www.nersc.gov/ [cited 13 May 2010]

76. TeraGrid. Home page. Available from: https://www.teragrid.org/ [cited 9 Feb 2010]

77. National Science Foundation. Award Abstract #0844219—Vulnerability Assessment of Grid Software Infrastructure. Available from: http://nsf.gov/awardsearch/showAward.do?AwardNumber=0844219 [cited 19 May 2010]

78. Basney, J. "MyProxy Protocol*." Global Grid Forum Experimental Document GFD-E.54*, 26 November 2005. Available from: http://www.ggf.org/documents/GFD.54.pdf [cited 19 Jan 2010]

79. Sakai. *Sakai Pilot Evaluation Final Report*. [PDF] 15 Oct 2009. University of North Carolina at Chapel Hill. Available from: www.unc.edu/sakaipilot/evaluation/FinalRept-Oct15-09-sm.pdf [cited 19 Jan 2010]

80. National Science Foundation. *Investing in America's Future: Strategic Plan FY 2006–2011*. September 2006. Available from: http://www.nsf.gov/pubs/2006/nsf0648/nsf0648.jsp [cited 19 Jan 2010]

81. Woods, D. Open Source Energy Savings. 2010. Available from: http://www.forbes.com/2010/03/01/energy-management-software-technology-cio-network-condor.html [cited 12 May 2010]

82. Brooks, F. P. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)* 1995: Addison-Wesley Professional.

83. Grubb, P. and A. A. Takang. *Software Maintenance: Concepts and Practice (2nd Edition)*. 2003, Singapore: World Scientific Publishing Co.

84. Software-Supportability.org. Home page. Available from: http://www.software-supportability.org/ [cited 22 Jan 2010]

85. SciDAC. Home page. Available from: http://www.scidac.gov/ [cited 13 May 2010]

86. NASA Goddard Space Flight Center. Reuse Readiness Levels (RRLs)—software reuse. Available from: http://www.esdswg.com/softwarereuse/Resources/rrls/ [cited 19 Jan 2010]

87. Association for Computing Machinery and IEEE Computer Society. *Computer Science Curriculum 2008: An Interim Revision of CS 2001*. 2008. Available from: http://www.acm.org//education/curricula/ComputerScience2008.pdf [cited 19 Jan 2010]

88. Carver, J. Software Engineering for Computational Science and Engineering. Available from: http://cs.ua.edu/~carver/Projects_CSE.htm [cited 19 Jan 2010]

89. Carver, J. "SE-CSE09: The Second International Workshop on Software Engineering for Computational Science and Engineering." In: *31st International Conference on Software Engineering (Companion Volume)*. p. 484–485. 2009. Vancouver, Canada.

90. Carver, J. "First International Workshop on Software Engineering for Computational Science and Engineering*." Computing in Science and Engineering*, March 2009. **11**(2): p. 8–11.

91. Kendall, R., D. Fisher, D. Henderson, J.C. Carver, A. Mark, D. Post, Rhoades, Jr., and S. Squires. "Development of a Weather Forecasting Code: A Case Study*." IEEE Software*, 2008. **25**(4): p. 7p. Vol. 25 Issue 4, p59–65 7p.

92. Basili, V. R., D. Cruzes, J. C. Carver, L. M. Hochstein, J. K. Hollingsworth, M. V. Zelkowitz, and F. Shull. "Understanding the High-Performance-Computing Community: A Software Engineer's Perspective*." IEEE Software*, 2008. **25**(4): p. 8p. Vol. 25 Issue 4, p. 29–36 8p.

93. Carver, J. "Post-Workshop Report for the Third International Workshop on Software Engineering for High Performance Computing Applications (SE-HPC07)*." ACM Software Engineering Notes*, September 2007. **32**(5): p. 38–43.

94. Carver, J., R. Kendall, S. Squires, and D. Post. "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies." In: *29th International Conference on Software Engineering*. p. 550–559. May 2007. Minneapolis, MN.

95. Kendall, R., D. Post, J. Carver, D. Henderson, and D. Fisher. *A Proposed Taxonomy for Software Development Risks for High-Performance Computing (HPC) Scientific/Engineering Applications*. Technical Note CMU/SEI–2006–TN–039. 2007. Software Engineering Institute, Carnegie Melon University.

96. Carver, J. "Third International Workshop on Software Engineering for High Performance Computing (HPC) Applications." In: *29th International Conference on Software Engineering (Companion Volume)*. p. 147. May 2007. Minneapolis, MN.

97. Carver, J., L. M. Hochstein, R. Kendall, T. Nakamura, M. V. Zelkowitz, V. R. Basili, and D. Post. "Observations about Software Development for High End Computing*." CTWatch Quarterly*, November 2006. **2**(4A): p. 33–37. (Invited Paper).

98. Hochstein, L. M., T. Nakamura, V. R. Basili, S. Asgari, M. V. Zelkowitz, J. K. Hollingsworth, F. Shull, J. Carver, M. Voelp, N. Zazworka, and P. Johnson. "Experiments to Understand HPC Time to Development*." CTWatch Quarterly*, November 2006. **2**(4A): p. 24–32. (Invited Paper).

99. Software Carpentry. Home page. Available from: http://software-carpentry.org/ [cited 19 Jan 2010]

100. Wilson, G. V. Software Carpentry: Essential Software Skills for Research Scientists. 2006. Available from: https://nanohub.org/resources/1811 [cited 29 Jan 2010]

101. Dye, C., "Every generation needs a new revolution: 200 years in the history of longevity*." SACEMA Quarterly*, November 2009(4). Available from: http://www.sacemaquarterly.com/docs/Dye-longevity.pdf [cited 19 Jan 2010]

102. Livermore Software Technology Corp. Home page. Available from: http://www.lstc.com/ [cited 19 Jan 2010]

103. National Cancer Institute. caBIG. Available from: https://cabig.nci.nih.gov/ [cited 13 May 2010]

104. iPlant Collaborative. Home page. Available from: http://iplantcollaborative.org/ [cited 13 May 2010]

105. Consortium for Ocean Leadership. Ocean Observatories Initiative. Available from: http://www.oceanleadership.org/programs-and-partnerships/ocean-observing/ooi/ [cited 19 Jan 2010]

106. Lynch, C. A., *Jim Gray's Fourth Paradigm and the Construction of the Scientific Record*, in *The Fourth Paradigm: Data-Intensive Scientific Discovery*, T. Hey, S. Tansley, and K. Tolle, Editors. 2009, Microsoft Research: Redmond, WA. p. 177–183. Available from: http://research.microsoft.com/en-us/collaboration/fourthparadigm/contents.aspx [cited 3 Feb 2010]

107. Lynch, C. A. "The Shape of the Scientific Article in the Developing Cyberinfrastructure*." CT Watch*, August 2007. **3**(3): p. 5–11.

108. Mesirov, J. "Accessible Reproducible Research*." Science*, January 2010: p. 415–416.

109. Plale, B., S. Miles, C. Goble, P. Missier, R. Barga, Y. Simmhan, J. Futrelle, R. McGrath, J. Myers, P. Paulson, S. Bowers, B. Ludaescher, N. Kwasnikowska, J. Van den Bussche, T. Ellkvist, J. Freire, and P. Groth. *The Open Provenance Model (v1.01)*. 2008. University of Southampton. (Unpublished)

110. Network for Computational Nanotechnology. Simuation, Education, and Community for Nanotechnology. Available from: http://nanohub.org/ [cited 19 Jan 2010]

111. ProgrammableWeb. Mashups, APIs, and the Web as Platform. Available from: http://www.programmableweb.com/ [cited 19 Jan 2010]

112. myExperiment. Home page. Available from: http://www.myexperiment.org/ [cited 19 Jan 2010]

113. Bala, V., E. Duesterwald, and S. Banerjia. "Dynamo: A Transparent Dynamic Optimization System." In: *SIGPLAN 2000 Conference on Programming Language Design and Implementation*p. 1–12. 2000.

114. Dehnert, J. C., B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson. "The Transmeta Code Morphing™ Software: using speculation, recovery, and adaptive retranslation to address real-life challenges." In: *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*. 37, p. 15–24. 2003. San Francisco, CA: IEEE Computer Society, Washington, D.C.

115. Adams, K. and O. Agesen. "A Comparison of Software and Hardware Techniques for x86 Virtualization." In: *ASPLOS 2006*. October 2006.

116. Amazon Web Services. Amazon Elastic Compute Cloud (Amazon EC2). Available from: http://aws.amazon.com/ec2/ [cited 28 Apr 2010]

117. VMware. Home page. Available from: http://www.vmware.com/ [cited 13 May 2010]

118. xen. Home page. Available from: http://www.xen.org/ [cited 13 May 2010]

119. Parallels. Home page. Available from: http://www.parallels.com/ [cited 13 May 2010]

120. QEMU. Home page. Available from: http://wiki.qemu.org/Main_Page [cited 13 May 2010]

121. Bochs. Home page. Available from: http://bochs.sourceforge.net/ [cited 13 May 2010]

122. Plex86. Home page. Available from: http://plex86.sourceforge.net/ [cited 13 May 2010]

123. Abramson, D., J. Jackson, S. Muthrasnallur, G. Neiger, G. Regnier, R. Sankarn, I. Schoinas, R. Uhlig, B. Vembu, and J. Weigert. "Intel Virtualization Technology for Directed I/O*." Intel Technology Journal*, 2006. **10**(3).

124. Woods, K. and G. Brown. "Assisted Emulation for Legacy Executables." In: *5th International Digital Curation Conference*. 2009.

125. Stewart, C.A. *Testimony before the United States House of Representatives Committee on Science and Technology Hearing on Leadership Under Challenge: Information Technology R&D in a Competitive World (2007 Report of the President's Council of Advisors on Science and Technology)*. 2008. Available from: http://hdl.handle.net/2022/5124 [cited 19 Jan 2010]

126. Edwards, P. N., S. J. Jackson, G. C. Bowker, and C. P. Knobel. Understanding Infrastructure: Dynamics, Tensions, and Design. January 2007. Available from: http://www.si.umich.edu/cyber-infrastructure/UnderstandingInfrastructure_FinalReport25jan07.pdf [cited 19 Jan 2010]

127. President's Information Technology Advisory Committee Panel on Open Source Software for High End Computing. *Developing Open Source Software to Advance High End Computing*. [PDF] 2000. Available from: http://www.nitrd.gov/pubs/pitac/pres-oss-11sep00.pdf

128. Fuchs, I. "Needed: an 'Educore' to Aid Collaboration*." The Chronicle of Higher Education*, 24 Sept 2004. **51**(5): p. B19. Available from: http://chronicle.com/article/Needed-an-Educore-to-Aid/13406/ [cited 19 Jan 2010]

Appendix 1: Position papers

Cyberinfrastructure Software and Sustainability Workshop
Index to Papers

March 26–27,2009

Open Source Development as a Pathway to Sustainability
Dick Repasky, Rich Knepper – Indiana University

Open Source Writ Large:
Advantages of a Foundation Community Model for Cyberinfrastructure
Rich Knepper, Dick Repasky – Indiana University

Sustainable Cyberinfrastructure Software: Perspectives and Priorities
CASC Paper
Submitted by Joel Replogle – Open Grid Forum

Sustainable Cyberinfrastructure Software for: DataAware Distributed Computing
Tevfik Kosar – Louisiana State University

Sustainable Software for Cyberinfrastructure
Reagan Moore, Arcot Rajasekar – University of North Carolina at Chapel Hill
Mike Wan, Wayne – Schroeder University of California, San Diego

Sustainable Software Ideas for the Next Generation Research Grid:
Standards, Interoperability and Software Development
David E. Hudak, Ph.D., Tom Bitterman, Ph.D., Neil Ludban, Stanley C. Ahalt, Ph.D. – Ohio Supercomputer Center

Sustaining Capabilities Not Codes By Architecting for Innovation
James D. Myers, Robert E. McGrath – National Center for Supercomputing Applications

Sustaining Software for Long-Lived Science Stakeholders of the Open Science Grid
Ruth Pordes – Open Science Grid Consortium

The Importance of Long-Term Science and Engineering Infrastructure for Digital Discovery
Nancy WilkinsDiehr – San Diego Supercomputer Center
Jay Alameda – National Center for Supercomputing Applications
Kelvin Droegemeier – University of Oklahoma
Dennis Gannon – Microsoft
Stuart Martin –Argonne National Laboratory

The iPlant Collaborative Position Paper
Richard Blevins – University of Arizona

A Framework for NSF Cyberinfrastructure Software Oversight

By: David A. Lifka (lifka@cac.cornell.edu)

Director, Cornell Center for Advanced Computing

Adj. Associate Professor, Computing and Information Science, Cornell University

Introduction

To become a national TeraGrid resource provider an institution must be able to demonstrate the intellectual merit and the broader impacts that will result from the availability of their proposed computational, visualization or data analysis resource. Intellectual merit for most TeraGrid resource providers to date has been related to providing resources of unprecedented computational scale in order to enable world class research never before possible. In order to accomplish this goal, budgets have predominantly focused more on systems acquisition and operations than on maximizing the availability of software, tools and new usage paradigms. As a result, the users of TeraGrid resources are a relatively modest-sized community of "super-scientists" that are not only leaders in their respective fields, but outstanding computational scientists and computer programmers. These super-scientists have made great breakthroughs over the past two decades due their high level of technical sophistication and the availability of national resources. Unfortunately, their level of computing expertise is not typical of many scientists and engineers in our country. A renewed focus by the national HPC community on software needs and new usage paradigms has the potential to bring the benefits of high-performance computing to many more scientists and engineers and, consequently, increase the impact of TeraGrid resources on U.S. scientific insight and discovery.

As the nation prepares for the TeraGrid "eXtreme Digital" future, establishing an independent, unbiased national software oversight committee focused on meeting the requirements of a much broader national science community could revitalize the nation's interest in computational science and enable researchers to secure maximum advantage from the world class resources to be deployed.

Critical Considerations

Software budgets are typically based on a resource provider's best attempt to provide a balanced solution for the general user community. As a result, common tools needed by many researchers are often not available on TeraGrid resources. The gating cost factors for software include:

Licensing Costs: Pricing models for commercial software are often prohibitive especially when applied to the scale of TeraGrid resources.

Support & Maintenance Costs: Both commercial and Open Source Software (OSS) solutions have ongoing software and maintenance costs. Relying on the TeraGrid community to provide support for specialized packages has significant staffing implications. Relying on the community for support and maintenance can produce mixed results in terms of timeliness, quality, and sustainability.

Development Costs: Funding the development of software is expensive in terms of direct cost or staff time and often fraught with risk. When funds are awarded for the installation of a large-scale TeraGrid resource, there are well-defined acceptance criteria with specific milestones and metrics of success that must be met before payment is made. The same level of detailed oversight is needed for software licensing/development awards.

Research Impact: Cost often is the primary factor when deciding what software will be available on a TeraGrid resource. Often this means that the best tool for a specific research domain is not available. This leads to lost opportunity or delayed discovery as researchers are impaired from doing their work in the most efficient manner and from leveraging the available national resources. Considering many researchers are funded by the NSF and other agencies, this lost opportunity cost, although difficult to measure, may be significant. Without a macroanalysis of software needs and a focused and efficient effort on meeting them, the "software gap" will remain unaddressed.

Sustainability: There are three strategies for sustainable software development. Funding with clear deliverables can make any of these strategies a viable option for delivering or developing the software needed by science and engineering researchers.

1**. Open Source Community Development:** This is popular in the TeraGrid community, but it requires significant subsidies from funding agencies. To ensure the satisfaction of the target user community, the decision process to fund these efforts needs to be transparent and include metrics of success and deliverable milestones based on user requirements in order to foster community confidence in the process.

2**. Support and Maintenance-Based Business Models:** Companies offering support and maintenance for community developed software has become a viable business model. Market competition and commercial incentives can produce a better user experience and, at the same time, sunset software that is inferior or no longer relevant.

3. **Commercialization:** To survive in today's market, the commercialization of a software package requires a very sound business model. The software must have broad applicability in order to create a volume market, or, it must be focused enough to provide a unique and superior capability for a niche market.

Ensuring Competitiveness: Competition is imperative if end users are to be well served. Careful consideration of software requirements, development deadlines, and the necessity of long-term maintenance and support is essential when deciding whether to fund a new software development project. Research institutions, open source communities, and commercial developers may all be viable competitors for providing software solutions depending upon the project scope, deliverables, and availability of funding.

Role of a National Software Oversight Committee

A National Software Oversight Committee should be established to: (1) investigate, analyze, and articulate the issues outlined in the previous section; (2) organize and conduct detailed studies on national research software needs; and, (3) recommend research software to be procured or developed by the following groups, including basic details on feature-functionality and need/availability time-lines:

- National Science Foundation
- Other Federal funding agencies
- TeraGrid XD management
- TeraGrid XD resource providers
- Research software developers
- Commercial software vendors
- Open Source Software developers
- Open Source Software service providers

Membership

Membership must include adequate representation of all stakeholders including funding agencies, a diversity of research disciplines, and advisors from commercial software vendors.

Executive Committee: 10-12 representatives, 1 per major research field or area of expertise including, but not limited to, engineering, physics, life sciences, social sciences, biology-medical, nanotechnology, computer science, and digital libraries. Members should be respected leaders in their fields with a solid understanding of state-of-the-art software and hardware systems (both commercial and open source) as applied to their field.

Ex-Officio Members: 1-2 representatives each from NSF-OCI, NIH, DOE, DoD-Modernization, and NASA

Advisory Members: Representatives from the software development and support industry as required by the executive committee. These representatives can provide realistic market and cost analyses that will be essential to the committee in order to generate sound guidance.

Terms and Funding

Executive Members will serve multi-year terms that will match the terms of TeraGrid XD initiatives.

NSF funding will be provided for:

• Travel: to attend meetings and conferences

• Effort: 1-2 months per year

• Materials & Supplies: Market analysis reports and other sources of information.

Focus Areas and Responsibility

The Oversight Committee members will be responsible for providing clear and detailed analyses of current state-of-the-art research methods with a focus on the shared and discipline-specific software requirements of a diverse national research community.

Current Software Availability: The Committee will maintain a web-based TeraGrid software index that details the software available on all TeraGrid resources by discipline and resource. Feature-functionality, limitations and support information will be provided for each software package.

User Requirement Analysis: The Committee will perform in-depth investigations of state-of-the-art software, tools and usage paradigms for a broad portfolio of research disciplines. Data will be collected, studied, and validated. Reports will be generated that provide data on the most popular software packages per discipline along with the associated costs, required or recommended platforms, and specific metrics that were used to rank one package over another.

Data sources will include, but not be limited to:

• Usage data and user requests from TeraGrid users and the management of TeraGrid resources

• Researchers from diverse scientific disciplines, who do not currently use TeraGrid resources or feel limited by the availability of software, the support for their usage model, or the level of readily-available consulting support required for them to make effective use of their application on a TeraGrid resource.

• Commercial and OSS software provider information, including feature functionality and licensing and maintenance costs.

• A user blog where users can share their thoughts, requirements and experiences in real time.

Recommendations and Requirements

Based upon budget availability, required feature-functionality, and the size of the user community, the Oversight Committee will make recommendations for the acquisition and support of various software libraries and tools. In the event that a critical software library or tool is missing, inadequate, or the licensing is cost prohibitive, the Committee will make recommendations to the various funding agencies to fund its development, support and maintenance. These recommendations will include detailed requirements, including:

• Compatibility, portability, scalability and integration with required security models

• Deliverable milestones, metrics of success, and evaluation procedures

• How best to foster competition to ensure breadth and diversity in the software development community and generate new innovations

• Sustainability models to ensure long-term availability of the software after the initial funding.

Benefits of a National Software Oversight Committee

- Engage authorities from diverse research communities to analyze the software needs and requirements for TeraGrid resources. Effectively meeting these needs will attract new classes of users to TeraGrid and broaden the scientific discovery impact of TeraGrid resources
- Development of clear requirements, specifications, milestones, and metrics of success to guide the entire
- software development community (engaging

research, open source, and commercial software expertise)

- Hard data, including costs, feature-functionality, scalability, interoperability, and security to enable the community to make well-informed software recommendations
- A transparent decision-making process for all TeraGrid software, including reasons for acquisition, development funding, or exclusion from the software portfolio
- Renewed software industry interest and competitiveness by providing unbiased cost and feature functionality analyses.

References

1. Investing in America's Future, Strategic Plan FY 2006-2011 (NSF-06-48).
2. Proposal and Award Policies and Procedures Guide, Part 1 - Proposal Preparation & Submission Guidelines, GPG. October 2008, Effective January 5, 2009 (NSF-09-01) OMB Control Number: 3145-0058
3. Cyberinfrasture Vision 21st Century Discovery, National Science Foundation Cyberinfrastructure Council, March 2007.

A view on scientific software sustainability

Ewa Deelman
Ann Chervenak
USC Information Science Institute

Science today is pushing the boundaries of hardware capacity and software capabilities. Applications such as those developed as part of the Southern California Earthquake Center (SCEC) project, the Laser Interferometer Gravitational-Wave Observatory (LIGO), and many others are making use of leadership-class clusters to simulate earthquakes, to search for gravitational waves, and to conduct other computationally and data-intensive analyses.

Through the NSF ITR program there were many scientific partnerships developed that fostered both computer science and domain science advances. In astronomy, new computational methods were developed to provide scalable solutions to science grade mosaic generation. In gravitational-wave physics, new data management techniques were developed to handle scientific data spread across millions of individual files. In earthquake science, new optimization techniques were developed to handle workflows with hundreds of thousands of individual tasks. As a result, new computational methods are supporting computational sciences in ways not possible earlier.

Although new, research-based, software is often the foundation of computational science advances, applications are sometimes reluctant to embrace new technologies. In some cases, applications are worried that there may not be funding for long term sustainability of software. They are reluctant to make their scientific progress dependent on software that may not be supported adequately for at least five to ten years. As a result, computational sciences may not get the benefit of the latest software innovations, and this situation may reduce the pace of scientific discoveries.

For the same reason, it is sometimes difficult for scientific software providers to attract new users and enhance software capabilities through the understanding of new requirements and challenges. Only through a partnership and close collaboration between software developers and scientists can advances be made both in computer science and domain sciences. One of the NSF programs that recognize the value of scientific software to domain sciences is the OCI SDCI program. It primarily funds development and maintenance of software that is used in a number of scientific disciplines. SDCI funds the Pegasus workflow management project ISI among others. OCI also supports the development and maintenance of Globus grid computing middleware at ISI. Although SDCI and other OCI programs provide funding for a number of software components that are used daily in a variety of scientific domains, it is not clear whether this funding will be sustained in the long run.

One possible approach to providing sustainability for scientific software is for applications and software providers to form partnerships and make the case to the funding agencies that continued maintenance, support, and development of software is necessary to maintain the pace of advancements across scientific domains. It is imperative that software development teams and scientists work together to justify the necessity of specific software components and to make those components more sustainable, since it is unrealistic to expect that funding agencies will indefinitely support the development and maintenance of software that has not been proven essential to scientific communities. Important responsibilities of software development groups include developing robust software based on good engineering practices and using existing commercial and open source tools as well as standards when appropriate. Such practices reduce the cost of software development and maintenance, making it easier for funding agencies to justify continuing support.

Some may argue that industry should take over scientific software and commercialize it. However, this model is often not possible. Scientific software is often continuously developed and enhanced as the understanding of application needs evolve, or as these need change over time because of the possible software enhancements. The software solutions also evolve over time because the problems that applications have are not straightforward and require research and experimentation. Thus, such software is typically not well suited to be handed off for production by industry. Often it is simply too expensive to commercialize the software given the relatively small number of potential users. Scientific applications are typically not a high volume, high-profit endeavor. Their value is more related to societal benefits rather than direct monetary profits, and thus often these applications need to be supported by government funding agencies for the benefit of national scientific competitiveness.

Today, funding agencies understand the importance of provisioning and maintaining high performance hardware and providing funds for the hardware support and maintenance. We believe that the same level of support should be given the scientific software that enables applications to run efficiently on that hardware.

Contact author: Ewa Deelman, deelman@isi.edu
3/1/2009

“After the Dance: The Hope for CI Software Sustainability”

Rion Dooley[1], Sudhakar Pamidighantam[2] on behalf of the Computational Chemistry Grid
1. Texas Advanced Computing Center, University of Texas at Austin, Texas
2. National Center for Supercomputing Applications,
University of Illinois at Urbana-Champaign, Urbana, Illinois.

February 20, 2009

The Computational Chemistry Grid (CCG, http://www.gridchem.org) was a NMI Deployment projected funded by the Office of Cyberinfrastructure (OCI) at the NSF. The primary contribution of the CCG was a computational chemistry oriented science gateway called GridChem. GridChem provides an intuitively familiar desktop portal to serve end-to-end requirements for computational chemistry communities. The GridChem client is deployed as a Java Web Start application while the GridChem middleware service leverages a mature service oriented architecture to aggregate complete control over a user’s VO under a set of common interfaces. Over the past year, GridChem has been among top Science Gateways in the TeraGrid in terms of total SU’s consumed burning over 500K SU’s in 2008 alone.

We are commenting as an organization that has developed and sustained a highly utilized, successful software project both inside and outside of NSF governance. We speak primarily from our experience developing user facing technologies and middleware that has been adopted and reused in support of other projects active today. In this paper, we share our experiences sustaining our software and conclude with recommendations for how to better sustain federally funded projects in the future. While our opinions very well may apply to simulation codes and HPC libraries, unless specified, we are not specifically referring to them.

The Blessing and the Curse

GridChem was born out of a need in the computational chemistry community for easier ways to conduct experiments using the most widely used software packages in the country. From the beginning of the project, our user base grew steadily. Because we had people using the software regularly, we were able to interact with them and adjust our development roadmap according to their emerging needs. As a result, were have benefitted from increasing adoption and utilization even beyond the end of the project.

We are now 6 months past the end of the project and our user count, number of jobs, and total consumed SUs all continue to rise. This is both a blessing and a curse. On the one hand we are pleased with the increased usage and adoption of our software. On the other hand, we do not have the funding to respond to many of the new opportunities to increase the reach of the software to new communities, applications domains and compute models.

Sustaining vs. Maintaining

One question we had to answer early on in the project was whether we wanted to sustain or maintain the software over time. Imagine taking your child to your family physician and him asking you if he should keep your child healthy or alive. It’s the difference between actively living and merely existing. We believe how one answers that question reveals a subtle, but distinct point about what one is trying to accomplish.

Webster’s Dictionary defines maintain as follows:

“to keep in an existing state: preserve from failure or decline”

Maintaining software, then, is the process of keeping it working. It does not speak to relevance. It does not speak of context. It simply implies that it continues to do perform the same task over time.

Webster's Dictionary defines sustain as follows:

"to give support or relief to."

Sustaining software, then, speaks of an ongoing commitment to keep the software relevant. It speaks of vitality. It implies that in times of change, support and/or relief are provided to help the software stay as relevant tomorrow as it is today.

In the context of GridChem, attempting to simply maintain the software would essentially be a death sentence. Our goal was never to maintain the project, but to create a sustainable tool that could remain relevant and useful over time. We wanted the software to have value to researchers and students in years to come even as the underlying technology changed.

To that end, we sought to leverage as much existing infrastructure as possible. Our initial focus was to adopt as much of the common national CI as possible. We soon discovered that there was not a generally adopted solution that could satisfy our most basic needs. Simple functionality to perform things like accounting, file, job and identity management, and resource monitoring were either not in place, did not exist, or did not perform well enough to leverage in a production setting. As a result, we spent considerable time in the first year filling in the gaps and creating the glue needed to put together a reliable, comprehensive middleware infrastructure.

We were not alone in our conclusion. We found this to be a common problem across may different projects. It seems that the developer community as a whole set out in parallel to solve these problems on their own time scales, for their own projects. As a result, there are several good solutions to many of these problems today. TeraGrid, OSG, OGCE and the DoE have done great works in providing stable reusable solutions for many of their needs. However, there is still no widely supported solution for what we feel are basic tasks. Additionally, it's important to note that multidiscipline and cross discipline activities are part of current science and engineering research. Things that we never felt were needs of the research community are now becoming expected capabilities to today's scientists. Engaging these emerging communities for when developing a national CI will become absolutely essential to the long term goals of the NSF.

Finding a Lifeline

As we stated, our goal was to sustain GridChem, not simply maintain it. In today's landscape there is no clear path for keeping good software sustained. With limited funds available at the national level, creating useful tools is not enough. Neither is creating highly utilized production solutions. Truthfully, those are both expected outcomes of any funded software project – and rightfully so. Good work is expected. Exceptional work is appreciated. The only real opportunity for GridChem to live on was through partnership and adoption.

The GridChem you see today is a project on its 3rd life. The original GridChem project was built on the ideas from an internally funded project at NCSA dubbed the Quantum Chemistry Remote Job Monitor. The formal GridChem project was funded through an NMI Deployment award from 2004-2007. In the Fall of 2007, the project was granted two no-cost extensions to run through the fall of 2009. During this time our focus shifted from aggressively increasing core functionality to hardening the middleware and client for increased production use, with limited funding. We also made a strong push during this time to increase awareness and evangelize GridChem to the community.

As of this writing, that work has resulted in two additional funding sources. The first is through inclusion on Cyber Resources and Facilities award out of the Chemistry Research Instrumentation and Facilities (CRIF) program at NSF. This is mainly to for small extensions to the client and middleware to support a slightly different execution model for the awardees. The second is through the TeraGrid Advanced User Services division to add much-requested parameterization and workflow support to the software.

A Value-base Culture

We believe that GridChem continues to thrive as a project because of the initial decision we made to sustain the software, rather than to maintain it. This attitude produced design and partnership decisions that resulted in a tool that has value to everyone who uses it. From the newest student to the seasoned researcher, GridChem meets them each at their needs. It enhances the process of conducting experiments thus allowing them to focus on their science rather than their computational needs.

In the end, the question of which software to support and who should do so almost seems too simple to answer. The mission of the NSF is clear.

"To promote the progress of science; to advance the national health, prosperity, and welfare; to secure the national defense...."

Software that meaningfully advances that cause will always have a place. Software that does not, will not.

Moving Forward

Having reflected on our experiences developing GridChem and participating in the developer community, we have several suggestions as NSF moves forward.

1. *Define and standardize a national CI*. Fund a permanent group to oversee this and provide high level training to the advocacy groups in EOT.

2. *Expose the national CI as a platform*. The development community can accelerate both the quality and adoption of technology if they can build upon a stable platform that exposes the underlying resources as core services as services. In doing so, developers can focus less on glue and adaptors and more on value-added services and tools that meet researchers at their existing comfort level using common interfaces.

3. *Build a self-propagating software feedback loop into future CFP*. This will create is a persistent review process for funded projects that will evaluate their place in the national CI.

4. *Fund hardening extension for projects showing promise and wide adoption*. Model the apache foundation's mentorship process to ensure best practices.

5. *Provide resources and funding to encourage community development*. Mature CI software requires a community to develop and mature the code. Creating this environment will lead to a stronger overall CI.

6. *Leverage the local expertise of MSI in stabilizing the national* CI and making them more useful to the general public. Incentivize this participation through awards of system time and collaborative extensions.

7. *Leverage virtualization to ease deployment and maximize ROI*. Providing personalized, or customized VO's for users and developers allows them to work from a position of stability. They know their sandbox is consistent with a well-configured production environment, so they can focus on their code rather than the environment.

8. *Aggressively encourage advance discussions on software interoperability and dissemination*. Including this in the CFP and the review process will encourage greater awareness of the national CI as well as better design in the early stages.

9. *Create a free flow of information between the global open-source community, industry, and academia*. Work with them to aid the tech transfer. Leverage local and state expertise and resources to make it happen. Extensible and interoperable CI development should be encouraged.

10. It is time to *plan for multidisciplinary science and engineering cyberinfrastructure* now to ensure maximum benefit for the research progress as all useful research is multidisciplinary in nature.

Corporate, Customer, and Academic Open Source Communities for Next Generation Software

Bryan Che, Red Hat
bche@redhat.com

Abstract

The NSF is looking both at models of how to build sustainable cyberinfrastructure software as well as specific software that will benefit its goals like providing communities with access to a "world class high performance computing (HPC) environment."[1] Red Hat Enterprise MRG, a high performance distributed computing platform which integrates Messaging, Realtime, and Grid capabilities, provides both an open source model of how academic researchers, customers, and corporations can collaborate as well as powerful software infrastructure which can help the NSF meet its next-generation cyberinfrastructure goals.

Introduction

Red Hat Enterprise MRG (Messaging, Realtime, Grid), is a platform for high performance distributed computing. It is innovative not only for the capabilities it provides but also for the collaborative and disruptive processes through which Red Hat has created and fostered this platform.

Each of the components in MRG—messaging, realtime, and grid—provides class-leading and powerful capabilities in and of itself. MRG Messaging provides messaging performance that is up to 100 times faster[2] than other messaging software and delivers many capabilities built into it that developers have traditionally have had to build on top of messaging software. MRG Realtime provides workloads on Linux extremely deterministic realtime performance and runs everything from command-and-control systems for US Navy warships to the fastest financial trading systems. MRG Grid provides a high-end scheduler that supports both traditional HPC workloads as well as utility or cloud models of computing; it can schedule and manage any workload, from sub-second calculations to server applications to HPC jobs across any computational resource, from local grids to remote grids to idle resources to virtual machines to private clouds to public clouds like Amazon EC2. Furthermore, the combination of Messaging, Realtime, and Grid technologies in MRG provide new capabilities like robust and scalable management infrastructure or low latency grid scheduling.

Beyond providing innovative capabilities for high performance distributed computing, however, Red Hat Enterprise MRG also provides good models of how NSF-funded researchers, end-users, and technology companies can collaborate to build sustainable cyberinfrastructure software.

Creating Open Standards and Open Source with Customers

MRG Messaging is open source software and also implements a new open messaging standard, Advanced Message Queuing Protocol (AMQP)[3]. When Red Hat first entered the messaging market, it was dominated by a handful of expensive and proprietary software products, and these products were highly specialized and uninteroperable due to the lack of a messaging protocol standard. This meant that many customers often bought multiple messaging products and de-

1 NSF'S CYBERINFRASTRUCTURE VISION FOR 21ST CENTURY DISCOVERY (http://www.nsf.gov/od/oci/CIv40.pdf)
2 http://www.redhat.com/mrg/messaging/features/#aio
3 http://amqp.org

ployed them in architectural silos, which was both expensive and complex.

Thus, when Red Hat began its work on messaging software, it did not just create an open source messaging implementation as this would not have addressed the core problem in the messaging space: even though messaging software is fundamental to distributed computing, the lack of an open standard for messaging was severely stunting the growth of a messaging ecosystem. Instead, Red Hat teamed with one of its customers, JP Morgan Chase (JPMC), to create an open protocol standard around messaging, AMQP.

JPMC, like many other banks, had developed its own messaging software to meet its high-end messaging requirements. However, JPMC had also written down the specification of its work, and this proved to be a good starting point for creating an open messaging protocol standard. Red Hat and JPMC created a legal contract to form the AMQP working group, which would develop this new standard as AMQP in an open and IPunencumbered manner. Then, they started bringing in many additional companies to collaborate in this working group.

AMQP now has strong participation from not just leading technology providers but also messaging end-users. The AMQP working group includes vendors like Red Hat, Cisco, and Microsoft to ensure good adoption and proliferation for AMQP-compliant products. But, it also includes high-end messaging users like JPMC, Goldman Sachs, Credit Suisse, Deutsche Börse, and Twist Innovations. Each of these users is actively contributing to the AMQP specification to ensure that AMQP will meet its needs for everything from performance to architecture to management. This ensures that AMQP will be a powerful but also practical standard.

This open and collaborative process between vendors and users has also extended to implementations of AMQP. Red Hat engineers started Apache Qpid[4], an open source implementation of AMQP and an upstream project for creating Red Hat Enterprise MRG. Just as the AMQP working group sees participation both from vendors and users, so the Qpid project has committers and contributors from both companies and customers.

The combination of an open standard like AMQP, customer-provider partnerships, and open source has led to a wide range of innovations in the marketplace. For example, there is now a hardware ecosystem developing around AMQP and the messaging space, so hardware devices will inter-operate with and complement messaging software with new capabilities. There is a robust software ecosystem building around AMQP. Many open source projects for things ranging from virtualization management to security infrastructure to Linux itself are incorporating AMQP. And, with both Red Hat and Microsoft's participation in AMQP, there is emerging true inter-operability between the Linux and Windows environments. All of this is fostered by a broad, cooperative community around open standards and open source.

Academic and Corporate Collaboration for HPC and Cloud Computing

MRG's Grid scheduler is based on Condor[5], a project created by the University of Wisconsin Madison and funded by the NSF. Condor powers Open Science Grid, an NSF-funded research grid, as well as many of the world's largest HPC grids. Red Hat has signed a strategic agreement with the University of Wisconsin (UW) around Condor which does two things: it made Condor open source under an OSI-approved license, and it established a development partnership between UW and Red Hat. For example, Red Hat has an engineering team on-campus at UW, working side-by-side with the Condor researchers there.

This partnership between UW and Red Hat is adding many innovative capabilities to Condor and also expanding significantly Condor's reach from research environments to enterprises. For example, Red Hat has focused on adding many capabilities which enterprises require for deployment but which are not paramount for academia. These enhancements range from new graphical management tools to enterprise

4 http://qpid.apache.org/
5 http://www.cs.wisc.edu/condor/

maintainability to concurrency limits on scarce resources like software licenses. Furthermore, Red Hat has also focused on advancing Condor towards utility and cloud models of computing by adding capabilities like libvirt virtualization support and Amazon EC2 integration. Many enterprises are now looking at MRG and Condor for building private clouds and moving to cloud computing.

Red Hat has also incorporated the AMQP messaging community into the Condor community. For example, Red Hat has developed an AMQP messaging-based job submission capability which enables Condor to schedule and execute sub-second calculations. Furthermore, the management infrastructure which Red Hat wrote for MRG is AMQP-based and provides an extremely scalable eventing-based architecture.

Beyond advancing Condor into the enterprise community, Red Hat's partnership and work around Condor is also feeding enterprise innovation back to the research community. For example, Red Hat has recently partnered with academic researchers to submit a proposal for implementing the next generation of TeraGrid[6], an NSFfunded research grid shared across multiple partner sites. This proposal will build upon open-source Red Hat Enterprise MRG and leverage capabilities like AMQP messaging and Condor grid scheduling for next generation HPC research.

Conclusion

Red Hat's collaborative work with customers, enterprises, and academic researchers in Red Hat Enterprise MRG has produced a class-defining, powerful platform for high performance distributed computing. Its robust and open capabilities around Messaging and Grid, and its Realtime performance deliver many of the requirements the NSF seeks around its cyberinfrastructure vision. Perhaps more importantly, though, Red Hat's open source collaborative approach to MRG has delivered new open standards for interoperability and ecosystems, has open sourced and brought NSF-funded grid technology to enterprises, has fed enterprise-led innovations back to academia, and has brought together organizations ranging from investment banks to public universities to technology companies to develop and benefit from these advancements together.

6 http://www.teragrid.org

Creating Sustainable, Coherent HPC Software

Jay Alameda
jalameda@ncsa.uiuc.edu

The software that the high performance computing community has come to depend on comes from a diverse range of sources, often tasked with solving problems that may not directly be related to high performance computing, and is assembled into unique combinations, not only on a per platform family basis, but also unique to each instantiation of a computing platform. We would like to suggest that some of goals that went into designing one essential (and now ubiquitous) component of high performance computing, MPI, should be considered in devising a strategy to raise the quality of the software environment being used in high performance computing. The hope is by considering these goals, and by understanding the processes by which current hpc software is developed and supported (including the funding models involved), we can not only increase the quality and consistency of HPC software across platforms and sites, but also improve the ability of our users and application developers to optimally use the HPC platforms being deployed across the nation, and indeed, the world.

The software that we depend on for our high performance computing endeavors is quite different from what is typically provided on a typical personal workstation. For instance, one vendor (eg, Microsoft, Apple) may be responsible for a large cross section of the system code that is running on this system; in this position, the single vendor can in a sense unilaterally declare how other codes (drivers, application codes, and so on) are encouraged and or allowed to interact with the operating system. In distinct contrast, HPC software often comes from a variety of sources, some customer driven, some community driven, some commercial, some open source, and is expected to support what quite often is a one-off combination of hardware assembled to provide a particular HPC platform. The burden of making this combination of software work properly is typically shared between a system integrator and a particular HPC site, in some combination of responsibilities. Upgrades to the software stack is typically fraught with peril, as the burden to prove that the more or less ad-hoc assemblage of parts continues to work as expected often shifts to the site providing the HPC capabilities, with some degree of assistance from the integrator. Validation tests and methodologies to assure that the software "stack" continues to perform as expected are rare and incomplete, leaving the burden of finding issues with the software assembly ultimately to the users.

As a result, it is somewhat surprising perhaps that we have been as successful as we have been in provisioning HPC resources for well over 20 years (for the NSF-funded supercomputing centers), with the large number of scientific breakthroughs that have occurred as a result of high performance computing through the years. In part it is due to the sheer determination of both system providers and the HPC sites; but one side consequence of these determined efforts to achieve success in the face of adversity is that systems and sites often function in a manner akin to feudal barons, which is not terribly friendly to users and less friendly to software developers trying to layer functionality on top of HPC platforms, including application ISV. The latter have retreated from time to time to the relative safety of desktop platforms, where the numbers of achievable licenses is large, and the software stack, well, is managed in a fairly coherent fashion.

In order to improve this situation, we would like to consider comments by Bill Gropp, in his recent Fernbach award address at SC08, when he was considering how to improve MPI. In order to consider the issue of improving MPI, Bill asked an important question:

"Is MPI the Least Common Demoniator Approach?" He asserted that indeed this is not the correct term to apply to MPI, rather, MPI sought to be the "Greatest Common Denominator". This distinction, argued Gropp, was critical as it changes how we make improvements. Something that is our "Least" common denominator leads to improvements by simply choosing a better approach – ie, it is hard to not do better than the "least". "Greatest" implies directly that improvements require *changing the rules*: either a) the available architecture support ("Denominator"), the scope ("Common"), or the goals (what is indeed "Greatest"?). Given this, Gropp argued that one can look for improvements, for MPI, for instance, in a some of the following ways. If one wants to change the common, for instance, by giving up on ubiquity or portability, one could more easily accommodate niches such as GPUs, FPGAs, etc. This could be good, but has potential costs such as placing one's efforts on the fringe, or falling off the commodity curve, or making changes in the wrong layer of abstraction (e.g., what impact should GPUs have on the message passing layer?)

If one wants to change the denominator, one needs to consider the base of features that are considered to be on every system. This may happen with new functionality being pioneered in the DARPA HPCS project, we will need to see if this shift comes to pass.

And finally, one can change the goals by expanding or contracting the meaning of "greatest". This could be in the area of distributed data structures, for instance, or support for concurrent activities – as a couple of ideas to consider for inclusion in the "greatest" contract.

So, with HPC software: can we identify the greatest common denominator? At the moment I believe we would be hard pressed to clearly identify the three components. As far as "greatest" is concerned, across a multitude of supercomputing sites there is little impetus to consider greatest. Indeed, in one project where commonality was sought after as an initial project goal (the NSF-funded distributed terascale facility, DTF), greatest unwound until the only requirement is a simple registration capability. All other components and capabilities are considered "optional". Granted, user demand for a number of the capabilities has made a large number of capabilities de-facto greatest, but the unwinding was spearheaded through an argument based on increasing architectural diversity as DTF morphed into a heterogeneous distributed capability.

As far as the denominator goes, can we inventory what features we assume are present on our HPC resources? Has this grown or shrunk as the field has progressed? Understanding our assumptions in this case can make our job easier for provisioning software, as being able to effectively use the common features will become more obvious as a design goal.

Finally, we need to have the discussion regarding what needs to be ubiquitous and portable across our diverse resources, and indeed how to handle unique and differentiating features. Handled correctly, we should be able to lower the amount of pain involved in moving between platforms – for application developers, for independent software (application) developers, and for developers of advanced tools that, for successful adoption of the tools to enable science and engineering, need to effectively work across a multitude of platforms with a minimum of "one-off" modifications to handle a class of platforms (or worse) – individual site modifications to an instance of a platform.

Not only do we need to consider these issues wrapped up in the "greatest common denominator" for HPC software, we need to consider how we fund such activities. Currently, HPC software is a highly leveraged activity in the broadest sense. Some activities are funded through relatively narrow, focused agency grants (e.g., the NSF SDCI program, which had one round of funding to date). Others are funded through the business model of an ISV, e.g., CFD codes, Computational Chemistry codes and so on; closely related are improvements to codes contractually obligated by ISV customers. Some components are off-shoots of research efforts, either in the application or enabling technology arenas. In either case, the transition from a research prototype to a robust software component is an arduous task that often is difficult to fund, not to mention technically difficult to achieve. Some components come from hardware vendors, as part of a strategic investment to help their core business (which quite often is not HPC!); others come from the open source community – which brings up another issue: who drives software maintenance and feature improvements. Looking across these diverse activities that bring HPC software to fruition – one can see that some codes are driven by the development community (such as open source software); some are driven by customers, and some are driven by research agendas. This results in software that is being pushed in directions that at times may be orthogonal to achieving optimal performance (in a broad sense of the word) in the HPC arena. *In order to improve the coherence of our HPC software activities, it is important to recognize the "crazy-quilt" of funding that keeps the activity going, to understand what the implications of this funding model are, and finally, to recognize that improving our state of the art in this field will require sustained and directed investment to not only address the gaps in funding, but to move to inject coherence across projects, platforms and sites.*

In conclusion, we have identified a number of shortcomings in our current models of provisioning HPC software and sites. We suggest a philosophical model or framework to motivate improvements to the state of HPC software, ie, the consideration of the "Greatest Common Denominator" across the HPC software space. Finally, we acknowledge that current software funding models are impeding progress in HPC software and that deliberate efforts to fund software aimed at raising the state of high performance computing is essential to providing a greatly improved environment for not only other software developers in the ecosystem, but for our ultimate customers: our end users.

Cyberinfrastructure Software Sustainability

Joshua Alexander
University of Oklahoma
jalexander@ou.edu
February 27, 2009

This position paper is in response to the call for papers from NSF for Cyberinfrastructure Software Sustainability & Reusability Workshop. The author of this paper takes the position of supporting open source community software through investments of time from developers and monies in the form of grants from government agencies. Without these investments, many different types of open source community software programs have become difficult to build or maintain. This paper purposes a simple solution to this problem: Pay developers to maintain, port, and test open source community software thru the resources of a local, regional, and/or national center.

The Current Landscape

Open source community software has been a staple of many of the applications High Performance Computing users utilize in their research. It is safe to say that without open source community software readily available for users to integrate into their applications, many advancements done with HPC research would not have happened in the time frame that they did. This availability of open source community software has fostered growth and expansion of many different applications and will continue to do so if and only if these codes are maintained and made portable to different platforms. Currently this is not entirely the case.

Some of the more popular open source software has matured over the years and become more streamline to build and maintain since many individuals made the initial investment of time and money. As a result, many of these software programs are now much easier to build than when they were first released and can be built on many different platforms due to the testing done by developers.

However, this scenario is only a small percentage of overall available open source software in the HPC community. The norm is much of the opposite. Many open source software programs are often difficult to build and maintain due in most part to a lack of investment in porting open source programs to different platforms. A good example of this is a piece of open source software named ECEPPAK.

The instruction of ECEPPAK state this program was been built and tested on two types of systems: An IBM AIX system and an SGI IRIX system. Since these systems are unique to their respective companies, many other HPC users cannot utilize this software without the help of either the developer or trained operational staff at a local, regional, or national center. Many times, the developer is no longer employed at the institution where the software originated and no one at the institution maintains the software there, so the software ends with the employment of the developer. Additionally, operational staff at a center either do not have the time or the training to port code to their system(s). This is an affective death blow to what could have been an otherwise popular piece of software among HPC users.

A Model to Follow

So what is the answer to keeping open source code alive, portable, and maintainable? A possible working model is the open source software GotoBLAS developed by Kazushige Goto of TACC. He has, according to the TACC FAQ website, developed "the fastest implementations of the Basic Linear Algebra Subroutines." Goto is employed by TACC, a regional HPC center and a part of TeraGrid, and continues to develop GotoBLAS for different platforms and architectures. While he is the sole developer of this software, many centers could use the same model and employ inhouse developers to both develop and maintain open source software. At the very least, give many developers accounts on their systems to allow for porting and testing of open source software.

As a result of Goto's hard work, many other open source software programs almost require the system have GotoBLAS installed in order to build their software. Since GotoBLAS gives the best performance, many open source developers recommend using GotoBLAS but still package a version of BLAS in their open source software. Software that still come with their own version of BLAS, may have been developed on systems that did not have GotoBLAS installed and therefore could not test with GotoBLAS but know GotoBLAS should give the best performance.

Summary

The sustainability of open source software can be achieved by funding developers to maintain and port open source software and by allowing developers access to many different platforms and systems located in local, regional, and national centers. The NSF can lead the way to software sustainability by using the model used by TACC.

Examining Preservation and Reconstruction of Existing Simulation Software To Understand Software Sustainability

Peter Bajcsy and Kenton McHenry, NCSA, UIUC

Abstract: This position paper reports on the examination of preservation and reconstruction of existing simulation software and summarizes the lessons-learnt from the perspective of long-term software sustainability. We have studied the problem of re-execution and reconstruction of the simulation software components used in the Crandon Mine decision process. The software, data and metadata needed by the software came from multiple agencies. Our study leads to a set of recommendations related to long-term software preservation and Cyberinfrastructure software sustainability and reusability.

Introduction: In the current digital era, two trends have been observed. First, there is an increasing amount of digital information that is generated, processed and reused in almost every aspect of our life. For example, decision making processes have become less paper based and more digital information based than ever before. Second, the underlying technologies supporting work with digital information have been changing more rapidly than any other technology [8]. These changes include not only computer hardware and operating system software but also data formats, networking capabilities, and application software (e.g., simulation software, data viewers or document editors). Following these two trends in the context of decision processes, there is a need to investigate how to deal with preservation and reconstruction of decision processes that are supported by Cyberinfrastructure including digital information, scientific software, and networking capabilities.

When it comes to preservation and reconstruction of decision processes, one finds himself at the crossroad of multiple types of records about the decision process. First, there are records prepared in the past and the corresponding decision processes that are being analyzed today. Second, there are records about decision processes that are being prepared today and will be analyzed in the future. Third, there are theoretical and experimental studies of how records could be prepared tomorrow so that the corresponding decision processes could be fully reconstructed in the future. Our work analyzes decision processes as being documented today with the goal of understanding how the decision processes could be documented in the future to improve preservation and reconstruction.

In general, we formulate the preservation and reconstruction problem using Cyberinfrastructure as follows. Given a computer assisted decision process that is represented by digital data, software and related documents, networking capabilities for accessing data repositories and computational resources, record management practices and the socioeconomic environment, we investigate the preservation and reconstruction methodologies that have to be in place in order to reconstitute the decision process later in time. This general definition of the problem incorporates the key research areas for information technology (software, scalable information infrastructure, high end computing, socioeconomic impact and management and implementation of federal information technology research) as identified by the PITAC committee [8].

The general problem is too complex to be solved directly and might be sub-divided into multiple sub-problems addressing challenges associated with preservation of only one component and one type of Cyberinfrastructure. Therefore, we narrowed down our focus to problems associated with only the reexecution of scientific simulation software, and explored the challenges of re-execution in the context of the Crandon Mine decision process.

Previous work: In the past decade, there has been an increased awareness of software preservation issues [5][6]. Communities of scientists and organizations have become concerned about preserving movies, video games and even web content [11][15]. For example, the Computer History Museum in Mountain View, CA, [9] has established a software preservation group[1] and started to define software selection criteria [10][11]. The Internet archive [12] is another organization that began providing archival services for the browsing of past web page contents using the way-back machine [13]. Some organizations, like the Software Preservation Society [14] started to specialize in the preservation of computer game software. The common goal of the above community and agency efforts, as well as our goal, is to establish preservation acceptance procedures for software of any kind so that the software can be re-executed in the future.

In the process of raising awareness, several basic and intriguing questions about software preservation have been posed: what socioeconomic or technical factors influenced the development and adoption of specific types of software[6], why preserve certain pieces of software such as computer games[1], and what should we save[5] [2]? Some of these questions and certain aspects of preservation could be addressed by going through reconstruction projects. The reconstruction of Charles Babbage's Difference Engine No. 2 in 2002 [3][4] with no contemporary original is such an example. Similar to [7], our work can be viewed as a reconstruction project where the goal is to understand the preservation and reconstruction requirements while attempting to re-execute simulation software supporting a particular decision process, such as the Crandon Mine decision process.

The problem studied and the outcomes: The problem is motivated by understanding the process of transferring digital information about decision processes from any federal agency to NARA. One such example is the case of the Crandon Mine Project [16] which spanned 24 years and involved multiple agencies and stakeholders. The project approval decision process required understanding of the environmental, ecological, hydrological, cultural and financial impacts. The decision makers from multiple domains obtained some understanding of the above by running simulation programs and analyzing their outputs. For example, there was a need to assess protection of groundwater resources. All simulation programs became one part of the decision process together with input data and parameter files. In our work, the goal is to explore a set of criteria for accepting simulation studies to the National Archives. The objective of the acceptance criteria is to enable reconstructions of these simulation studies in the future from the digital information preserved today.

The term "reconstruction" here refers to the re-execution of the Crandon Mine Project simulation software with the associated meta-data and input data set to recreate the stored outputs. The simulation utilized several applications. ANNIE, a user interface designed and developed to assist the user in all aspects of hydrologic modeling and analyses, could be executed and interacted with. However, it was not clear what interactive inputs were used to setup the simulation parameters (.wdm file). The key simulation software, HSPF, was not available in the archives we had. With regards to data, files were selected for the reconstruction according to the user manual (files with extensions uci, wdm, plt, exs, and ech). Using the data was difficult as we encountered inconsistencies in file naming and incompleteness in the input data. For example, there were missing wdm, plt, and/or exs files. In a nutshell, although we found subsets of required data files they did not meet the requirement of the software during execution.

Preliminary Recommendations: Base on our Crandon Mine specific reconstruction effort, we concluded that at least the following steps have to take place before a successful re-execution of the simulation software could be achieved:

- Verify input and output files by testing with included software for possible format corruption
- Include a batch file or a shell script for setting up the environment variables
- Document the platforms and data that the software was tested with.
- Define acceptable differences between provided output files and the output files obtained by software re-execution.

1 See http://community.computerhistory.org/scc

Further, before preservation the software should be re-executed by starting with a bare bones system, and following the installation and execution instructions to reproduce results. It would be beneficial to establish forms specific to software acceptance that would provide an inventory of record context and content, as well as a software execution trail (e.g., last accessed, last executed). It has been suggested that the use of forensic tools [17] could be used to recover some of this information. It is apparent that filling the software context and content forms might be an extra burden for agencies using software, managing software versions and preserving the software. We foresee software sustainability and reusability in automating the information gathering that would have to be entered otherwise into the software context and content forms. The automation could be accomplished by designing self-describing workflow environments such as Cyberintegrator[2] where the data sets, software and executions are automatically annotated. While there are many perspectives on the problem of software sustainability and several approaches, there is a need for complete information in terms of binary executables, inputs, outputs to compare to, and adequate documentation to allow for future re-execution.

References

[1] H. Lowood, "Playing History with Games: Steps towards Historical Archives of Computer Gaming," the Electronic Media GroupAnnual Meeting of the American Institute for Conservation of Historic and Artistic Works, Portland, Oregon, June 14, 2004

[2] L. Shustek, "What Should We Collect to Preserve the History of Software?, IEEE Annals of the History of Computing Published by the IEEE Computer Society, October –December 2006., pp. 110-112

[3] Doron D. Swade, "Historical Reconstructions," IEEE Annals of the History of Computing ,vol. 27, no. 3, p. 3, July-September, 2005.

[4] D. Swade, "Collecting Software: Preserving Information in an Object-Centred Culture," History of Computing: Software Issues, Springer, 2002, pp.232-235.

[5] M. Campbell-Kelly, "Software Preservation: Accumulation and Simulation," IEEE Annals of the History of Computing, vol. 24, no. 1, 2002, pp.95-96.

[6] James W. Cortada, "Researching the History of Software from the 1960s," IEEE Annals of the History of Computing, vol. 24, no. 1, pp. 72-79, Jan-Mar, 2002.

[7] R. Moore, J. F. Jaja and R. Chadduck, "Mitigating Risk of Data Loss in Preservation Environments," Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'05), 11-14 April 2005 Page(s):39 – 48

[8] "Information Technology Research: Investing in Our Future, PITAC (President's Information Technology Advisory Committee) Report," Feb. 24, 1999, National Coordination Office for CIC, URL: http://www.nitrd.gov/pitac/report/pitac_report.pdf.

[9] Software Preservation at the Computer History Museum, URL: http://www.computerhistory.org/

[10] Interview with Sellam Ismail, Curator of Software ," Software Preservation at the Computer History Museum," The DoD Software Tech, URL: http://www.softwaretechnews.com/stn8-3/ismail.html

[11] In the "The Attic & the Parlor: A Workshop on Software Collection, Preservation & Access" Computer History Museum, Mountain View, CA May 5, 2006 , URL: http://community.computerhistory.org/scc/workshop/CHM_Attic_Parlor_Workshop_Proceedings_May-05-063.pdf

[12] The Internet Archive, URL: http://www.archive.org/about/about.php

[13] The Internet Archive – Wayback machine, URL: http://www.archive.org/web/web.php

[14] The Classic Software Preservation Project (CLASP), URL: http://www.archive.org/details/clasp

[15] Su-Shing Chen, "The Paradox of Digital Preservation," Computer 2001, pp. 24-28.

[16] The History of the Crandon Mine Project, URL: http://dnr.wi.gov/org/es/science/crandon/

[17] John J. L., Adapting Existing Technologies for Digitally Archiving Personal Lives, iPRES, 2008.

2 http://isda.ncsa.uiuc.edu/cyberintegrator/

FEDORA PROJECT, Fedora Electronic Lab in Research & Development environment

Chitlesh Goorah, Fedora Electronic Lab Architect
Email: chitlesh@fedoraproject.org

ABSTRACT

This paper entails how the Fedora Project encourages R&D in advanced electronics design through its Fedora Electronic Laboratory (FEL) platform. Fedora has opted a different approach in the development of such high-end hardware design and simulation platform. This approach focuses mainly on providing opensource EDA solutions to meet several high-end design flows and methodologies, rather than the traditional opensource method: random packaging process.

INTRODUCTION

There are many opensource Electronic Design Automation tools on the internet. These EDA tools include advanced scientific know-how of the semiconductor industry, brought by many people from various countries. In order to design hardware for the real life, these EDA software should fit into a particular design flow and ensure interoperability. Otherwise, this knowledge is useless for real life situations. On the other hand, this scientific knowledge is incorporated under various software languages. Thereby, it is also very difficult for the end-user to maintain such design flows, along with an EDA Consortium compatible operating system.

FEL strives to lighten the work load of the electronic design community who should only care about their cutting edge technologies rather than wasting time on software deployment. Hence, the Fedora Project incorporates and prepares those opensource EDA tools into its EDA portfolio that meets the current Semiconductor industry's trend along with an EDA Consortium compatible operating system, Fedora.

Fedora Project models a Semiconductor design center as its user base to provide the best portfolio for Micro-Nano Electronic Engineering design flows with opensource software. With such a model, both the Semiconductor industry and R&D facilities have for free and without any registration full access to this electronic engineering portfolio. The latter covers the following fields:

- Verilog code generation.
- Analog/Digital ASIC design and circuit simulations.
- Modeling, PCB design and HDL synthesis.
- Post tape-out chip testing facilities.
- Embedded Systems Development.
- Verification means for hardware description languages (VHDL and Verilog).
- Standard cell libraries supporting up to a feature size of 0.13 _m.
- Extracted spice decks which can be simulated with any spice simulators.
- Interoperability between various packages in order to achieve different design flows.

RESEARCH & DEVELOPMENT ENVIRONMENT

- The Research & Development environment (either academic or industry oriented) has established strong ties with the major EDA vendors. These EDA vendors have resources to improve their software in terms of

- runtime and advanced algorithms (routing, retiming, parasitic capacitance calculations)
- inclusion of a new industry standard
- characterized standard cell libraries of different technologies

Despite those advantages, the end-user experiences 'Vendor Lock Down' whenever the design project requires interfacing with a simulator from a different vendor or even home-made scripts. FEL strives to provide the appropriate design tools which support industry standard data formats in order to eliminate the so-called 'Vendor Lock Down' issue.

Although the Fedora Project is a non-profit organization, FEL shares the same difficulties the big EDA vendors are facing. While many EDA vendors also use opensource tools (GNU Toolchain, Java, Tcl/Tk, Perl,...) to build their software, Fedora as the linux distribution is a perfect ground for their development, since it is an EDA Consortium compatible OS. The researcher thereby has a better design experience by cohosting opensource EDA tools and commercial EDA tools. Hence, the home-made and project dependent scripts created by the researcher do not suffer from obsolete optimizations and functions of different compilers or interpreters.

Since the last two years, Fedora Electronic Lab has been improved to support hardware design for the following applications:

- Academic and Research
- Automobile
- Defense/Space
- Medical

The user-base is thus expanded to fit different researchers' needs. While the design and simulation tools proposed under the FEL umbrella are used to design vending machines, RISC processors, consumer products, thick film circuits and different types of pixels, FEL is a reliable option for R&D department to consider.

Researchers can use FEL as a testbench for their hardware algorithms. Since the EDA software is opensource, researchers don't have to rewrite the whole simulator in order to verify their specific hardware algorithms. They can also use Fedora as a collaborative development platform with other researchers of different institutions with Fedora's VOIP solutions. Researchers can share, backup and retrieve their measurement data securely together with opensource version control systems.

AN OPENSOURCE EDA COMMUNITY BUILDER

FEL was spinned from an opensource software community with the same values of the Fedora Project. It bridges both opensource hardware and opensource software communities. Subsequently, maintaining a community around FEL is subjected to deal with culture clashes between the communities being bridged.

Our approach with FEL is to build a community around opensource EDA solutions with opensource software. This differs from the old-fashioned opensource software community's method which proposes only opensource EDA software. This old-fashioned method downplays the value of the opensource EDA community as it does not interface with the field applications. This defeats the purpose of being an EDA provider.

Our approach with FEL though is to find first EDA solutions from field-proven problems. These EDA solutions will eventually lead to a proper set of opensource EDA software that meets the demands of the field-proven problems. Researchers not only receive EDA portfolios from FEL but also a community that can listen to their real-life problems with EDA software.

- To build such a community both inside and outside FEL requires a robust infrastructure as a model to follow. Such a model should help the FEL developer-base to fix all the mentioned bottlenecks and to focus on FEL's roadmap alone. Fedora's current infrastructure serves FEL with many facilities in terms of
  - marketing
  - packaging maintenance and release
  - software optimizations and security

The only problem is the targeting userbase is not the same as regular Fedora userbase. FEL's userbase may not visit linux tradeshows and thus the value of the opensource EDA solutions are not properly conveyed to the right userbase. Since FEL holds the responsibilities of doing the marketing of its upstream projects, new ways need to be found. That said, it is imperative to line up FEL marketing with those of big EDA vendors. One of the opensource software community clashes FEL ignores is competition with other linux distributions. Hence we have a clear roadmap for our milestones which in return also helps the userbase achieving the design sign-offs quickly.

CONCLUSION

Fedora Electronic Lab has proved that advanced electronic design is possible with opensource EDA software. While its userbase focuses on "technology-to-product" transition, the Fedora Project continues to sustain both the opensource EDA community and EDA solutions for the best electronic design experience. The EDA portfolio supported by Fedora Electronic Lab will help its users to achieve their goals. This new approach of EDA software deployment was well greeted by several academic institutions in India, France and United States. The next release Fedora 11 Electronic Lab will push the quality barrier higher than the previous releases.

ABOUT FEDORA ELECTRONIC LAB

Fedora's Electronic Lab[1] is a sub-project of the Fedora Project[2] dedicated to support the innovation and development of opensource EDA community. This ambitious sub-project provides a complete electronic laboratory setup with reliable open source design tools in order to keep engineers and researchers in pace with current technological race.

REFERENCES

[1] http://chitlesh.fedorapeople.org/FEL

[2] http://www.fedoraproject.org

Fedora Project – UAF Geophysical Institute

Dr. Jef Spaleta

This paper summarizes my personal thoughts on how the Fedora Project can serve as a model for the NSF to follow to sustain the development and widespread use of a diverse ecosystem of scientific software in an open and transparent manner. The NSF's role would parallel the one which Red Hat currently plays in the Fedora Project, by making key investments and setting project direction, while leaving room for community to grow the project in new ways. I've been selected to write this paper from the Fedora perspective in part because, as a scientific researcher and Fedora contributor, I straddle both the science and software development communities. In the following discussion I focus on two aspects of the Fedora project structure I feel are most relevant to the CI Software Sustainability Workshop goals and NSF's core mission.

The Fedora Project

The Fedora Project is a global partnership of free software community members, managed by Red Hat, which makes key infrastructure and resource investments to build collaborative space and incubate innovative new technologies, some of which may later be integrated into Red Hat products. These technologies are developed in Fedora's collaborative space and produced under an open source license from inception to allow their study, adoption and modification by other projects. The largest of the many free software creations of the Fedora Project is the Fedora distribution, a Linux based operating system that provides users with access to the latest free and open source software, in a stable, secure and easy to manage form. The Fedora distribution itself is composed of many individual software components, each with their own developer communities outside of the direct control of the Fedora Project or Red Hat. The Fedora Project is structured to act as a conduit between the needs of software users and the interests of software developers, where both groups benefit from the integration efforts that produce the Fedora distribution.

The Fedora Project is more than just software, though. It is a community of contributors with a variety of skill sets, who work to advance the interests of the free culture movement. It's never enough to produce the better technology. There is a need to engage and educate users of that technology to efficiently spread the benefits of innovation. The inclusive nature of the Fedora Project aids immensely as it gives non-computer programmers a way to directly impact the success of the project to educate and to put the technology innovations into the hands of a growing number of people. Just as a science literate society can make better use of science innovation, we find that a society literate in the open development process can make better use of the open innovation being produced.

All the components of the Fedora distribution and the Fedora Project's infrastructure services are open and made available under OSI approved licensing. This is done to ensure that Fedora always remains free for anybody, anywhere, to use, modify and distribute. This is important to the collaborative process, both internally to Fedora and externally for projects that Fedora draws from. The Fedora Project is composed of hundreds of individual contributors, and draws on the work of thousands of developers from individual open source projects. The adherence to OSI approved licensing makes it possible for people to build on the work of others while limiting the complications associated with licensing restrictions. Public sector institutions, corporate interests and private individuals can all contribute to extending the value and build on each others work because they all have equal access to use the technologies the Fedora Project provides.

Private-public collaborative partnerships, such as cutting-edge science and technology research, that rely on common software to form the basis of that collaboration become more difficult and more costly when the software licensing terms discriminate between category of collaborators.

This focus on open, collaborative development makes Fedora a center for innovation in free and open source software. The Fedora community contributes everything it builds back to the free and open source world and continues to make advances of significance to the broader community, as evidenced by the regular and rapid incorporation of its features into other Linux distributions. Regardless of which Linux distribution you use, you are relying on code developed within the Fedora Project. The NSF could choose no better goal than to take as strong a leadership role in collaborative science innovation as the Fedora project takes in the open collaborative software ecosystem. The key aspects of the Fedora Project which make it a successful partnership between community and business interests, its open collaborative nature, could equally benefit the NSF's mission to act as a catalyst for scientific innovation which maximizes the return on investment to the American public.

Fedora as a Model for the NSF

There are some strong parallels between the Fedora project and the NSF's mission as expressed in the four NSF principle foundations of discovery, learning, infrastructure and stewardship. Ultimately Fedora aims to grow and sustain the ecosystem of open source development, driving forward software innovation from which everyone can benefit and on which they can continually build. The NSF's mission is to similarly advance science innovation. The NSF can and should use the Fedora Project as a blueprint for policy and infrastructure, and as a starting point for an NSF managed initiative. Although the NSF goal of managing resources for highly collaborative and innovative science will undoubtedly present challenges foreign to the Fedora Project itself, the Fedora Project's approach to solving challenges should still be very relevant.

Open Infrastructure Investments Targeting Enhanced Collaboration

The most critical aspect of the Fedora Project's success is openly developed infrastructure services that aid in all stages of the open source software development process. Red Hat invests into Fedora infrastructure, which lowers the barrier on collaboration for the entire Fedora community. Some of that infrastructure is used in building and distributing Fedora software, but the services maintained by the Project also include a raft of collaborative hosting services at Fedora Hosted (http://fedorahosted.org) for software development not necessarily tied to the distribution itself. Fedora Hosted provides hosting and support services for individual software projects, which lowers the burden on individuals starting a new development effort. In addition, the Fedora project maintains infrastructure for communication such as mailing lists, collaborative documentation (wiki), and a Voice-over-IP (VoIP) service.

These infrastructure investments lower the barriers towards collaboration, but the fact that they are open source projects also encourages individual contributors to extend these services as new needs are identified. For example, the Fedora translation community has built and integrated a new translation service, Transifex, to better meet their own needs. This work has even spawned a separate private business effort, independent of Red Hat, to further the development of the translation technology (http://www.indifex.com).

The success enjoyed by the Fedora Project clearly demonstrates the need for the NSF to invest heavily in infrastructure that facilitates collaboration for scientific researchers and the sharing of data, tools, and results. An open high performance computing infrastructure is one area in which targeted NSF investments could effectively mimic the Fedora Project approach, and even potentially coordinate with the software development ongoing in Fedora. Another potential area of significant overlap is the establishment of an NSF coordinated collection of open data visualization and analysis tools that encourages the establishment of a extensible, best of breed data analysis toolkit.

There are however other challenges unique to the NSF's mission with no existing Fedora infrastructure parallel, but in which an emphasis on lowering barriers to collaboration still applies. One such challenge is data archiving. The NSF may need to invest in a long-term, centralized data hosting repository, and to develop a set of client policies for data submission and retrieval. This repository would lower the barrier to collaborative science using NSF funded datasets, and ensure the accessibility of data beyond the funding horizon of the initial collection effort. Different virtual observatory initiatives could then be built encompassing subsets of the information stored in this repository. Such a repository could host primary data, and associated analysis results with appropriate attribution histories accounting for each step of the collaborative process for use in publications and other communications. This repository may be able to leverage the growing experience with distributed version control systems in use now in the open source software ecosystem, which produce authoritative attribution records for highly collaborative work.

Managing Competition and Diversity

To a large extent, Fedora does not attempt to micromanage what individual contributors work on in an effort to pick winning technologies. Because of the way Fedora is structured, the project is able to accommodate a wide diversity of software at most levels of the Fedora distribution. Most community member work on what they want to work on for whatever reasons they choose, and individuals software components die out or flourish based on interest. Good ideas cross-pollinate at the cost of having multiple implementations that duplicate effort to some extent. A key set of Fedora staff keep the processes moving and the collaborative infrastructure operating, but for most part contributors work on the pieces of technology they find most compelling, whether its for personal reasons or they are being paid by their employer to do the work.

Over time, the NSF may develop a preference for one technology over another to build on when choosing to fund additional tool development or to integrate tools into a cohesive framework. What software components to fund directly can be a complicated assessment involving an understanding of project momentum and overall health. Even if such a preference is established, when possible, it is appropriate to leave room in the shared infrastructure for competing ideas or alternative implementations to exist even if they are not a funding priority. Alternative approaches can be fertile ground for new ideas, even when code reuse is preferred.

But its not always the case that different, competing software projects can work side-by-side in an integrated way. For these cases Fedora has developed a featuring process associated with the timebased releases which let us evaluate how and when major changes are introduced. A committee of elected Fedora community members review the feature requests for applicability and impact. If the NSF were to commit to an integrated software deliverable such and analysis toolkit or a HPC reference platform the feature review used in the Fedora distribution release process might apply to the NSF effort.

Summary

In summary, I believe that Fedora Project makes for a worthwhile reference model for organizing a sustainable, highly collaborative project that incorporates diverse community interests and pays dividends for both public and private sector contributors. I have so far highlighted only two aspects of the Fedora Project structure that the NSF could use as a blueprint in its own efforts to manage a sustainable ecosystem for scientific software development . There are other aspects of the Fedora Project, such as the governance model and the time-based release cycle, that the NSF could adapt to serve its own goals but would require additional discussion beyond the space provided for here.

Long-term Sustainability for Cyberinfrastructure Software: Lessons from the Library Community

MacKenzie Smith, Associate Director for Technology, MIT Libraries
February, 2009

Introduction

The U.S. cyberinfrastructure initiative and its predecessors have produced a wealth of new software and related technologies for large-scale e-science research, but their long term sustainability and value is still unclear. Much of this software is expensive to design, produce, support, maintain, and use, and adoption can be very slow due to a variety of factors including complexity of use and difficulty reaching and supporting the tens of thousands of individual researchers who are potential adopters.

Software sustainability includes at least three dimensions: initial development and ongoing innovation; maintenance and support over time; outreach and successful deployment to the intended audience of scientists and engineers. If any one of these dimensions is missing or fails then the entire endeavor fails, but funding today tends to focus only on the first dimension and not enough planning or resources are applied to the others. In fact, the "last mile" problem of reaching individual scientists and engineers with innovative technology appears to be the biggest stumbling block to long-term sustainability.

These dimensions apply to software that is centrally managed or distributed, visible to researcher end users or not. Even software that is centralized and invisible to its end users needs to have measurable value to those users to attract the resources required for ongoing support and maintenance. However the different dimensions work best in different modes. For example, software development can be either centralized or decentralized, outreach and support to end users is best done locally, but community management and support is best achieved with a centralized model. The solution to cyberinfrastructure sustainability needs to recognize the hybrid and flexible nature of the overall system, and fund the distinct parts differently.

The Example of Institutional Repositories

Starting in the year 2000, research universities began producing and deploying "institutional repository" enterprise systems to capture, describe, manage, and curate digital research output (primarily research publications but also including multimedia and research data assets). Institutional repositories, like much cyberinfrastructure, combine off-the-shelf components (databases, search engines, markup languages and processing tools, Webbased data visualization tools, etc.) with features unique to the research enterprise. Since the common software is typically maintained by other sectors with similar needs and more resources – finance, energy, transportation – the design goal for institutional repositories was to minimize the amount of software that requires ongoing maintenance by the research community.

The major institutional repository platforms – currently DSpace, Fedora, and EPrints – are completely open source software and have achieved a remarkable degree of adoption in the past six years. Almost every research university or other research-generating organization in the world runs one of these repositories, typically at the research library with support from the institution's IT department.

What caused the broad adoption of these systems in the research community?

Ensuring support for scientific and engineering community standards (e.g. for data encoding); brand recognition and confidence in the software producers;

ease-of-use and availability of support, training, and documentation of the software; ability to customize and improve software as necessary; ability to switch to newer products over time. But the most important factor was identifying exactly the “market” for the infrastructure and how to reach it, and putting more resources into the outreach and support than the original software development by several orders of magnitude.

The sustainability strategy for institutional repositories addresses all three dimensions described above:

• For DSpace and Fedora, small, independent and non-profit foundations were created to support the user community. These foundations provide coordination of development and maintenance work throughout the user community and represent the community in various settings, e.g. policy efforts, standards groups, external communities of supporting software such as database and search engine software.

• New software development and innovation is funded with new money, typically by an institution that uses the software but not necessary the original developer. In some cases, groups of institutional users form to jointly develop new software (e.g. for a new version or major new feature).

• Ongoing user support and maintenance are provided by the user community so that the cost and risk are distributed widely.

• End user outreach and support is provided by the software users (typically research library staff) who also measure the use of and value to the end users of the software, to justify ongoing investment.

Recommendations to the NSF

The model of small, lightweight, independent organizations to manage software communities has proven effective for some components of cyberinfrastructure. Since such organizations are typically not the source of software development or maintenance itself, a sustainable ecology of stakeholders can develop that include commercial entities contributing to the overall health of the community. Judicious open source software licensing can make this easier to encourage. The organization coordinates activities throughout the community to insure that institutions who wish to innovate can reach a target audience to test the idea and get help with and adoption of the new software.

If the scope of such organizations is broadened to include other distributed and reusable components of cyberinfrastructure; if these organizations directly support the individual institutions that, in their turn, support individual researchers, this would create a flexible organization to sustain cyberinfrastructure software indefinitely into the future. Since such organizations could represent a wide range of software systems and wouldn’t depend on software sales or government funding for operating revenue, they could avoid over-zealous promotion of solutions that don’t meet researchers’ needs. In fact their survival would depend on their effectiveness at meeting researchers’ needs over time, since they would be measured by service to scientists rather than software development.

Finally, such organizations would be able to market the cyberinfrastructure software to sectors outside scientific research with common needs (e.g. financial, energy, transportation) to create an even richer ecosystem to support the software going forward. For all major existing cyberinfrastructure that has achieved true sustainability (e.g. the Internet and the Web) this cross-sector utility was clearly a key to success.

NSF funding should focus on new development and innovation, but coordinated with the community management organizations to discourage the phenomenon of reinventing the wheel. Additional, separate funding could support outreach and training to adopting institutions and to end user researchers. By decoupling these two, we would hope to achieve both continued innovation and improved adoption and sustainability of at least some of that work.

Open Source Development as a Pathway to Sustainability

Dick Repasky and Rich Knepper
Pervasive Technology Institute, Indiana University, Bloomington, IN, USA

There are multiple paths to sustainability, and no one path fits all projects. To match paths with projects one must understand for each path to sustainability the conditions under which it will succeed as well as those under which it will fail and then ascertain which types of projects will flourish on that path. In this position paper, we open a discussion of sustainability through the open source method of software development by reviewing literature on the method and by asking which types of funded projects are amenable to it.

Open source is as much of a method of software development as it is a type of license, and it is the development process that must be sustainable. Briefly, software is developed by a distributed group of loosely organized volunteers, usually without financial compensation. The method has produced some innovative, highly successful software, and it has piqued the interests of both economists (e.g., Lerner and Tirole, 2002) and organizational scientists (e.g., von Hippel and von Krogh, 2003), who ask why it works and how it fits into the conceptual frameworks of economic theory and organization science, respectively. Two issues emerge: why participate, and what are the sources of leadership and vision that keep projects focused? People participate because they need the software and are willing to invest in it, because they wish to gain experience to enhance career prospects, because they wish to flaunt skills to attract potential employers or investors, and because they enjoy the work. Leadership and vision usually come from project founders or small groups of senior developers. To be sustainable, projects must attract a sufficient pool of developers and retain developers long enough to maintain continuity.

For cyberinfrastructure projects, we assert that the primary motivating factor that will sustain open source projects is need for the product and willingness to invest in it. Case studies clearly indicate that need and willingness to invest have been sufficient to motivate some projects. However, it is not yet possible to evaluate objectively the suitability of open source development for individual projects because bounds on the conditions under which need and willingness to invest are sufficient to sustain projects have not been explored.

We also assert that a major hurdle to be overcome is the transition from being a supported project to an independent project. We discuss two aspects of the transition: maintaining the community of developers and the cessation of external support.

The ideal transition is one in which the community of developers is unperturbed, a transition that is possible only if projects embrace open source development very early. The later that projects move to open source development, the greater the change that the development community undergoes, and the greater the chance that the community disintegrates. The most extreme transition is one in which the original development team abandons a project, say when funding runs out, and a new open source team must form to sustain the project.

Which types of cyberinfrastructure projects are amenable to open source development early in life? We assert that the open source path to sustainability is better suited to infrastructure projects than it is to research projects. The goal of infrastructure projects is to create something useful that will indeed be used, and there is no reason to delay the process of building a distributed community of developers. Indeed, infrastructure projects should benefit from the efforts. The goals of research projects on the other hand are to demonstrate concepts and the abilities of researchers. Researchers tend to be secretive about their work, especially the source code, until they have mined it for publications, and then when they have finished with it, they are ready to abandon the work and move on to something else. That is, they are prone to open up their work to the open source software development process only very late, under conditions that we believe are least favorable to sustainability through open source. Pathways to sustainability other than open source are probably better suited to research projects. Funding agencies that wish to pursue open source for research projects might pursue it through grants that are awarded after research has been completed, with the understanding that only the concepts will have been demonstrated and that codes will need to be rewritten from scratch because researchers routinely sacrifice principles of software engineering for speed in demonstrating concepts.

The second aspect of the transition from a supported project to a project that is sustained through open source development is the change from being sustained by funding to being sustained by contributions of effort. The questions are: what conditions are necessary for independence and what further development do funding agencies want to support? From the viewpoint of a project more funding for longer periods of time should always be better because the project should be able to produce more functionality sooner than with less funding. Agencies that seek to maximize the amount of cyberinfrastructure produced, on the other hand, are likely to favor early independence. We view funding decisions as policy decisions to be made by the agencies. Nevertheless, we encourage agencies to develop clear policies and guidelines regarding their criteria for independence and the cessation of support for projects that are to be sustained by open source development.

References

Lerner, J., and J. Tirole. 2002. Some simple economics of open source. Journal of Industrial Economics 50(2):197-234.

von Hippel, E., and G. von Krogh. 2003. Open source software and the "private-collective" innovation model: issues for organization science. Organization Science 14(2):209-223.

Open Source Writ Large: Advantages of a Foundation Community Model for Cyberinfrastructure

Rich Knepper and Dick Repasky
Pervasive Technology Institute, Indiana University, Bloomington, IN, USA

Open Source software (OSS) presents a number of unique benefits to the Cyberinfrastructure (CI) community. These include include low costs for software distribution after initial development, ability to alter software to fit new requirements, accessibility of source code even after projects end, and a collaborative model of development. In addition, OSS is adaptable in that code can be changed to meet new requirements, even at a different institution by a different development team, and the source code is not subject to intellectual property restrictions that would make it unavailable when a vendor company fails or is acquired by another company. We identify two types of community development around OSS projects: the project centered community and the foundation community. The project centered community is the traditional model of OSS development in which developers and users participate in a single project. The project leads (usually one person or a small group) attempts to attract users and developers in an effort to build a community centered around that project. As a result, the environment of OSS projects developed in this way varies significantly. Foundation community development creates groups of projects around an area of interest (for example the Apache Foundation is centered around "providing web services"), providing a framework around which new projects enter the foundation via an acceptance process and potential participants have a better understanding of how projects work based on their familiarity with other projects in the foundation. For the NSF, using the foundation community model for OSS development may result in increasing the number of sustainableCI software initiatives. We propose that the foundation model for community development may have substantial benefits for the development of CI software, and note that some of the characteristics of the foundation community model require forethought and engagement on the part of those who would make use of this type of software development.

Open Source Software can specifically benefit the CI community in a number of ways. The NSF funds software development and purchases for CI and as a government agency it is accountable for reducing costs as much as possible; sponsoring the creation of OSS allows for the NSF to budget for development and maintenance, rather than relying on vendors' promises that licensing fees will not change and reduce lockin to vendor licenses. Additionally, the highly specialized nature of Cyberinfrastructure software projects often means that contracting or purchasing software is prohibitively expensive. The collaborative nature of OSS development meshes with that of the CI community, in which both projects and software development is geographically and institutionally distributed, decision making and communication is asynchronous, and partnerships and confederations are the rule. Finally, different versions of the same software can be developed concurrently and exchanged between teams to fit varying needs at different institutions, and pieces of code can be exchanged and reused for different projects that have similar needs.

The foundation community is demonstrated best by foundations such as the Apache Foundation, the Mozilla Foundation, and the Free Software Foundation. The Apache Software Foundation (ASF) was created in order to provide software for web services. The ASF is a loosely governed body that determines which projects are included and provides a common set of rules and member roles, operational infrastructure, and principles for development (described as "The Apache Way"): collaborative software development; commercial friendly standard license ; consistently high quality software; respectful, honest, technical based interaction; faithful implementation of standards; security as a mandatory feature[1]. The foundation prides itself on functioning as a meritocracy—potential members with access to change the source code must demonstrate both ability and conformance to the Foundation's principles—and as a "do-ocracy" initiatives are pursued by those willing to carry them out and coordination is achieved via consensus gathering.

This foundation community model provides a number of benefits over the project centered community. Most importantly, several characteristics of the foundation community create stable expectations for potential participants. A foundation for open source software typically defines a standard license or set of licenses for software to conform to, a common problem or issue to address (web services in the case of the ASF, tools for an operating system in the case of the Free Software Foundation), governance structures and organizational culture that all members accept. The foundation and its projects are known quantities and various projects operating under a given foundation may be expected to have similar means of communication, dispute resolution, and requirements for membership. Some foundations provide physical infrastructure for projects. By making use of a common infrastructure and reducing redundant systems, projects that are part of foundations can take advantage of economies of scale. Finally, a foundation such as the ASF becomes a known entity for potential contributors. A project that is part of the Apache project carries some cachet by working under the aegis of a larger project that is well known and accepted as an accomplished and effective project. Furthermore, organizations that establish themselves have inertia. While individual projects may come and go, the foundation becomes an institution, and users, developers, and sponsors all have set expectations about the projects associated with it.

Cyberinfrastructure development projects can benefit specifically from the foundation community model of OSS development. CI projects using this model are more likely to be sustainable because of the stability that is ensured by interacting with a foundation rather than a population of loosely related projects. By sponsoring or working with foundations that address a given problem or issue, CI organizations are able to identify and address specific questions outside the general OSS project population. Finally, foundations create entities separate from individual projects that can exhibit considerable longevity. When projects are no longer relevant, the areas of interest that they address often remain and developers and users can move to new projects handled by the overarching foundation with relatively little difficulty, rather than seeking new suppliers for software.

The NSF has a number of routes to encourage the establishment of foundations for Open Source CI software. NSF can actively seek to create (via funding initiatives) foundations in order to address specific questions or issues. In doing this, it is important to understand that the basic principles of the organization (such as "The Apache Way") are as crucial to establish as goals and metrics for assessment —which means that there are requirements of the means of software development as well as ends, for sustainability's sake. The establishment of a foundation, determination of governance structure, creation of boards, decisions on basic licenses and principles requires considerable engagement on the part of the funding agency, if this is the route to be taken. The question or issue of interest ("web services", "tools for an operating system") may have substantial influence on the effectiveness of the organization. The ASF has a central problem that is rather tightly defined, and certainly of interest to multiple parties, some of which are prepared to contribute considerable resources to the area of interest. In contrast, the central problem of the Free Software Foundation is somewhat more diffuse and of necessity broadens the scope of the organization and the entities involved in projects.

Another route to encourage sustainable development of CI software would be to contribute to individual projects which are part of an existing foundation, or to stipulate membership in a foundation as a condition of funding. This reduces considerably the amount of engagement required on the part of the granting agency in comparison to creating a wholly new foundation, while retaining benefits conferred by foundation membership. Furthermore, if proposal authors are able to secure a letter of commitment from an existing software foundation, this smooths the proposal pro-

1 Apache Software Foundation. "How the ASF Works." http://apache.org/foundation/howitworks.html accessed 2/19/2009.

cess and both the proposers and the granting agency have some assurances about the quality and expected longevity of the software for development.

We believe that the foundation community model can contribute to the sustainability of CI software by providing the benefits of establishing a known entity (longevity, name recognition and notoriety), setting stable expectations, economies of scale, and concentration of similar efforts under an umbrella organization. It is essential to note that successful foundations have significant cultural components and that a considerable amount of thought needs to go into those components, if the organization is to be truly sustainable. In addition, the area of interest is also of crucial importance to the scope and boundaries of the organization and the contributors which it can consider attracting.

Sustainable cyberinfrastructure software: perspectives and priorities.

23 March, 2009

The Coalition for Academic Scientific Computation (CASC – http://www.casc.org/) is an educational nonprofit 501(c) (3) organization with more than 50 member institutions. CASC members represent many of the nation's most forward-thinking universities and computing centers dedicated to enabling and supporting the use of advanced cyberinfrastructure to accelerate scientific discovery for national competitiveness. CASC institutions contribute to medical discoveries and healthcare, global security, economic vitality, and the development of a diverse and well-prepared 21st century workforce.

CASC herein presents two recommendations regarding steps that should be implemented so that sustainable software for cyberinfrastructure and computational science can be incubated, developed, published and supported to best serve the NSF mission and U.S. national interests.

The mission of NSF can be described as "…promoting achievement and progress in science and engineering and enhancing the potential for research and education to contribute to the Nation" [1]. Inarguably, the transfer of research outcomes and their supporting technologies to both the broader research community and toward economic development opportunities is a major contribution to the nation. Among all of the areas of science supported by NSF, high performance computing (HPC) is acknowledged as a key factor in accelerating research [2] as well as economic development [3]. HPC can reduce the time required to gain scientific insight into a problem, help to design new drugs and medical devices, simulate a process that is otherwise impossible to understand or model, design new consumer products, and lead to the discovery of new processes and phenomenon.

Sustainable cyberinfrastructure (CI) software is a critical enabler of HPC for both research and economic development [4]. Unfortunately, we observe that much of cyberinfrastructure and computational science software developed under NSF-funded projects has limited impact beyond fundamental research. By their very nature, the cyberinfrastructure and computational science research areas routinely generate software.

However, this software is utilized by only a small number of individuals – typically the principal investigator, a small group of students and perhaps a few collaborators. Consequently, we believe that much of the value of that software is never harvested and the potential benefits for both research communities and for economic development are never realized.

There are many reasons for this state of affairs. For example, many researchers seek answers to specific scientific questions, and the code they develop is a means to that end. In such instances, software is considered as "disposable" rather than a "deliverable." Second, many researchers, including very 'expert' HPC users and researchers, are not trained in software engineering techniques. Third, many research codes are only expected to be used a relatively few times, and taking the time to modify these codes so that they are reasonably easy to understand is generally considered orthogonal to the primary line of scientific inquiry. Fourth, taking the time to transition code from "user-hostile" to "user-friendly" takes considerable time, usually requires extensive testing, and is, in many cases, not funded under current NSF practices. Fifth, in some cases, NSF supports custom software development when commercial applications are available (and in certain instances, superior), but the licensing terms (costs, proprietary code, etc.) for commercial software prohibit adoption by research and education institutions. Finally, while there are numerous open-source software repositories, for example Source-Forge [5], the Apache Software Foundation [6], Code Haus [7], and Google Code [8], the cyberinfrastructure software development community has not broadly adopted the use of such repositories, at least in part due to overall maturity of the software and various licensing issues. Despite these hindrances, the NIH has recently demonstrated an ongoing commitment to software developed under NIH funding [9] and is reaping the benefits of that investment.

The Council on Competitiveness (CoC) has recognized the importance of HPC in the private sector with its HPC initiative since 2004, focusing on gaining an understanding of how HPC can be used across the private sector to drive productivity and competitiveness. In July 2005, the CoC and Ohio Supercomputer Center (OSC) hosted a workshop on the "Evaporation of the HPC Application Software Market," encouraging independent software vendors, public and private sector HPC users, HPC vendors, and public sector funders of HPC R&D to create a framework for action to stimulate the creation of needed HPC application software [10]. Participants discussed the challenges of maintaining and creating HPC application software suitable for a competitive, corporate "production" environment, the state of the ISV application software market, and the role of government, universities and national laboratories to help accelerate development of new and/or updated code. Subsequently, a CoC/DARPA "Study of Independent Software Vendors (ISVs) Serving the High Performance Computing Market" found that the business model for HPC application software is fast disappearing, as there is an increasing need for expensive, long-term development of highly scalable codes [11]. While industry has shown enthusiasm in adopting cloud computing and high throughput models, the lack of scalable application software is preventing more aggressive use of HPC by industry, and seriously impeding industrial and national competitiveness.

The programs sponsored by NSF regularly generate research software in every area of science and engineering. In order to reap greater benefit from this software, NSF sustainable cyberinfrastructure software policies should be created or re-evaluated in terms of transfer and sustainability to the broader research community as well as for economic development purposes. In doing so the NSF should avoid the temptation of implementing unfunded mandates to its grantees or propose models that do not reflect a realistic long-term approach to software sustainability.

Two complementary efforts should be considered. First, the research community must be engaged and encouraged to produce "transition ready" software. Second, crossdisciplinary software efforts must be undertaken to discover, develop, harden, and adapt some research codes into production codes for broader scientific use and for economic development.

Recommendation #1: the cyberinfrastructure and computational science research community must be engaged and encouraged to produce "transitionready" software. The term "transition-ready" software is meant to refer to software that is ready to be handed to the broader community, perhaps in nominally useful form, so that the software can be preserved in order to consider it for further development effort. Transition-ready software will likely lack the robustness, user interface and portability of production software, but should represent a buildable, testable, adequately formatted and documented software project. For example, transition-ready could include an associated set of manual pages that allows for meta-data that is associated with the code. In the open source community this is commonly done by embedding comments in the code and then using automated tools to extract the comments in order to form rudimentary manuals. Using JavaDoc [12] for example, comments are extracted, massaged, and semi-automated standardized document pages are produced from comments that were manually inserted into the code. When the software pages are generated in html they can, in turn, be accessed across the web during discovery processes launched to determine what codes are available to perform a particular task, and where those codes are made available. We note that this approach is simplistic, and will not work on extremely large codes that are sometimes developed by the HPC community.

We believe a logical approach for the NSF is to engage the cyberinfrastructure and computational science research community in creating transition-ready software, since the original research team must perform the early steps of research code transfer. We acknowledge the difficulty of asking researchers to take on software transition in addition to their primary research mission, but only the original research team understands the discipline, the algorithm, the implementation and the success criteria of the research code.

Additional software transition funding could be provided as part of an individual program's solicitation and proposal review, provided that adequate mechanisms could be developed for proposing and reviewing development plans. This funding could be provided on an optional basis, providing the opportunity for the research team to apply for supplemental funding to create transition ready software. Additionally, cross-disciplinary programs could be created for proposing and evaluating software transition projects. Further, the NSF should also consider funding for tool and standards development for the cyberinfrastructure and computational science community to develop transition-ready software. Such efforts could include training on software development and code expectations. Finally, the NSF should also consider funding a full program dedicated to software maintenance, following the example set by NIH.

Recommendation #2: the NSF should support a modest number of sustainable software development teams and a national repository of HPC software. Crossdisciplinary software efforts must be undertaken to discover, develop, harden, and adapt select research codes into production code for broader scientific use and for economic development. These interdisciplinary software development teams should be focused on both discovering and developing new, public domain, highly scalable codes that will have broad scientific and engineering impact.

In particular, we feel that the scientific and industrial communities would benefit greatly from: (1) a repository of non-classified, nonproprietary, open source, scalable codes developed using new and existing federally funded software and algorithms - a good example of a parallel effort, albeit closed, in the DoD can be found in the CREATE program [13] sponsored by the High Performance Computing Modernization Program Office; (2) integrated test and validation tools that can be uniformly applied to software sustained in the repository and a comprehensive suite of published benchmarks that can be applied to any emerging architecture that will also promote good code design; (3) software transfer support to mature successful research codes for transition to the private sector; and (4) outreach teams to work with academia and industry to identify key HPC software gaps which have not been addressed adequately by either party and are critically important to solving challenging problems, or to meeting contemporary business needs.

To begin this process, we recommend conducting a national CI review and assessment. Pending the results of the study, the NSF should work in partnership with other federal research agencies as well as key players in the open source and commercial domains, to establish a "best practices" standard of what constitutes National CI quality code. Further, the team(s) that springs from this review should establish and serve as gatekeeper to a national code repository, ensuring that existing gaps are filled and a cohesive national CI suite is maintained. At times the teams will have to do the hardening of code when the original developer is not available. Other times they will oversee the hardening process. In the end, accountability to the standard must be held by developers and the team(s) alike to ensure that the national CI moves forward as a whole.

We suggest that funding for the above recommendations be provided in three ways. First, the NSF should build significant software funding into the budgets of grants to resource providers or competitively selected CI software development teams working in partnership with regional centers or resource providers, to develop software that can be successfully deployed. This will guarantee that the solutions produced are hardened, tested, and production-ready. It also produces an environment where natural selection of the best tools will prevail and remain up-to-date. Second, funds for a hardening of tech transfer should be allocated as necessary for an initial period of three years, with reduced ongoing funding. Lastly, the EOT budget should be increased and utilized to educate researchers on the established "best practices."

The benefits of these recommendations will be felt in the first- and second-tier centers, as well as minority serving institutions. The lessons learned and the tools produced from the competitive review process will reduce redundancy and development time, thereby increasing the overall science output of the community by keeping researchers focused more on their science and less on their computational software.

NSF recognition and support for such these activities is directly in line with the NSF mission and would have a dramatic impact on the success rate of cyberinfrastructure and US computational research as a whole.

References

[1] The NSF Mission, http://www.nsf.gov/nsf/nsfpubs/straplan/mission.htm NSF Technical Report.

[2] Simulation Based Engineering Science: Revolutionizing Engineering Science through Simulation., NSF Technical Report. http://www.nsf.gov/pubs/reports/sbes_final_report.pdf.

[3] Report to the President of the US, Computational Science: Ensuring America's competitiveness, President's Information Technology Advisory Committee, June 2005 http://www.nitrd.gov/pitac/reports/20050609 computational/computational.pdf

[4] NSF's cyberinfrastructure vision for 21st century discovery, NSF Cyberinfrastructure Council Technical Report, 7.1, July 20, 2006. http://www.nsf.gov/od/oci/ci-v7.pdf

[5] Source-Forge, http://sourceforge.net/

[6] Apache Software Foundation, http://www.apache.orgsourceforge.net/

[7] Code Haus, http://www.codehaus.org

[8] Google Code, http://code.google.com

[9] http://grants.nih.gov/grants/guide/pa-files/PAR-08-010.html

[10] Council on Competiveness and the Ohio Supercomputer Center, "Accelerating Innovation for Competitive Advantage: The Need for Better HPC Application Software Solutions." http://www.compete.org/publications/detail/383

[11] Council on Competiveness, IDC, DARPA, "Study of ISVs Serving the High Performance Computing Market: The Need for Better Application Software, Council on Competitiveness, IDC, and Defense Advanced Research Projects Agency." http://www.compete.org/publications/detail/393

[12] Java-Doc, http://java.sun.com/j2se/javadoc/

[13] High Performance Computing Modernization Office 2007 Annual Report: http://www.hpcmo.hpc.mil/Htdocs/DOCUMENTS/Annual_Report_2007.pdfhttp://www.hpcmo.hpc.mil/Htdocs/DOCUMENTS/Annual_Report_2007.pdf

This white paper has been approved by a majority vote of the current members of CASC (with no votes opposing endorsement of this white paper recorded as of time of submission). Contributing authors: R. Dooley, J. C. Facelli, J. Gemmill, D. E. Hudak, K. L. Kelley, D. Lifka, J. Odegard, S. C. Ahalt, , and other CASC member representatives.

This paper may be cited as: Dooley, R., et. al., "Sustainable cyberinfrastructure software: perspectives and priorities," Coalition for Academic Scientific Computation (www.casc.org).

Sustainable Cyberinfrastructure Software for: Data-Aware Distributed Computing

Tevfik Kosar, Louisiana State University

Scientific applications and experiments are becoming increasingly complex and more demanding in terms of computational and data requirements. Large experiments, such as high-energy physics simulations, genome mapping, and climate modeling generate data volumes reaching hundreds of terabytes per year. Data collected from remote sensors and satellites, dynamic data-driven applications, digital libraries and preservations are also producing extremely large datasets for real-time or offline processing. To organize and analyze these data, scientists are turning to distributed resources owned by collaborating parties or national facilities to provide the computing power and storage capacity needed. But the use of distributed resources imposes new challenges. Even simply sharing and disseminating subsets of the data to the scientists' home institutions is difficult and not yet routine — the systems managing these resources must provide robust scheduling and allocation of storage resources, as well as efficient and reliable management of data movement.

Although through the use of distributed resources the institutions and organizations gain access to the resources needed for their large-scale applications, complex middleware is required to orchestrate the use of these compute, storage, and network resources between collaborating parties, and to manage the end-to-end processing of data. The majority of existing research and development efforts has been on the management of compute tasks and resources, as they are widely considered to be the most expensive. But, as the famous quote attributed to Seymour Cray *"A supercomputer is a device for turning computebound problems into I/O-bound problems"* states, the management of data resources and data flow between the storage and compute resources is now becoming the main bottleneck for especially large-scale data-intensive applications.

Traditional distributed computing systems closely couple data handling and computation. They consider data resources as second class entities, and access to data as a side effect of computation. Data placement (i.e., access, retrieval, and/or movement of data) is either embedded in the computation and causes the computation to delay, or is performed by simple techniques which do not provide the same privileges as compute jobs. The inadequacy of traditional distributed computing systems in dealing with complex data handling problem in our new data-rich world requires a new paradigm called **data-aware distributed computing.**

In this new paradigm, data placement activities should be represented as full-featured jobs in the end-to-end workflows, and they should be queued, managed, scheduled, and optimized via specialized data-aware schedulers. As part of this new paradigm, a set of new tools should be developed for mitigating the data bottleneck in distributed computing systems, which will provide capabilities such as planning, scheduling, resource reservation, job execution, and error recovery for data movement tasks; integration of these capabilities to the other layers in distributed computing such as workflow planning, resource allocation, and storage management; and optimization of data movement tasks via dynamically tuning of underlying protocol transfer parameters.

Most important software to sustain and maintain in support of the NSF mission:
NSFs 'Cyberinfrastructure Vision for 21st Century' states *"The national data framework must provide*

for reliable preservation, access, analysis, interoperability, and data movement [3]." The data-aware distributed computing paradigm will especially address these important issues, and advances in this area promise to enable a wide range of new high-impact applications and capabilities, which is closely aligned with the NSF report on 'Research Challenges in Distributed Computing Systems' [2].

Similarly, the DOE Office of Science report on 'Data Management Challenges' says *"Although many mechanisms exist for data transfer, research and development is still required to create schedulers and planners for storage space allocation and the transfer of data [4]."* And, according to the 'Strategic Plan for the US Climate Change Science Program (CCSP)', one of the main objectives of the future research programs should be *"Enhancing the data management infrastructure"*, since *"The users should be able to focus their attention on the information content of the data, rather than how to discover, access, and use it [1]."* This statement by CCSP summarizes the ultimate goal of many cyberinfrastructure efforts initiated by NSF, DOE and other federal agencies, as well the research direction of several leading academic institutions.

We believe that the "data–aware distributed computing" paradigm will be a big step forward to reach this goal. It will not only impact computer science research by changing the way computing is performed, but it will also dramatically change how domain scientists perform their research by facilitating rapid analysis and sharing of raw data and results. Future applications will be able to rely on this new transformative paradigm to manage data movement and storage reliably, efficiently and transparently. The impacted application areas will include all traditionally compute intensive disciplines from science and engineering, as well as new emerging computational areas in the arts, humanities, business and education which need to deal with increasingly large amounts of data.

Early Examples of Data-aware Distributed Computing:

One of the earliest examples of data-aware distributed computing is the **Stork** data scheduler (www.stork-project.org). Stork implements techniques specific to queuing, scheduling, and optimization of data placement jobs, and provides a level of abstraction between the user applications and the underlying data transfer and storage resources. Stork introduced the concept that the data placement activities in a distributed computing environment need to be first class entities just like computational jobs. Later, this novel idea was also acknowledged by the strategic reports of federal agencies. The DOE Office of Science report on 'Data Management Challenges' defined data movement and efficient access to data as two key foundations of scientific data management technology [4]. The DOE report also said: *"In the same way that the load register instruction is the most basic operation provided by a CPU, so is the placement of data on a storage device... It is therefore essential that at al l levels data placement tasks be treated in the same way computing tasks are treated"* and referred to the Stork data scheduler [5].

Another project based on the same idea is the **PetaShare** distributed data archival, analysis and visualization system (www.petashare.org). PetaShare storage network links nine Louisiana research institutions, leveraging 40 Gbps Louisiana Optical Network Initiative (LONI) infrastructure to make the interconnections and fully exploiting high bandwidth low latency optical network technologies. PetaShare makes use of data-aware storage and scheduling technologies to transparently and efficiently enable more than fifty senior researchers and two hundred graduate and undergraduate research students from ten different disciplines to perform multidisciplinary research. Application areas supported by PetaShare include coastal and environmental modeling, geospatial analysis, bioinformatics, medical imaging, fluid dynamics, petroleum engineering, numerical relativity, and high energy physics.

Last year, researchers from around the world came together at the first international workshop on data-aware distributed computing **(DADC'08)** to discuss this new computing paradigm and its impact on large-scale complex applications (www.cct.lsu.edu/~kosar/dadc08). DADC'08 was held in conjunction with HPDC'08 and explored especially the problems in data aware scheduling, resource allocation, metadata collection, workflow management, and visualization. The distributed computing and data management communities joined forces in an effort to generate productive conversations on the planning, management, and scheduling of data handling tasks and data storage resources. The second DADC workshop will be held this year as part of HPDC'09.

Recommendation:

In its 'Cyberinfrastructure Vision for 21st Century', NSF already acknowledges the importance of technologies for reliable and efficient data movement, access, and analysis. We believe that NSF should give very high priority to support and maintain software in these areas since *"In the future, U.S. international leadership in science and engineering will increasingly depend upon our ability to leverage this reservoir of scientific data captured in digital form [3]."* The advancement in data-aware distributed computing will capitalize NSF's investments on TeraGrid, DataNets and other large-scale cyberinfrastracture and computational science efforts; and will directly impact scientific iscovery and economic development in the nation. It will greatly strengthen a broad range of research, engineering, and development activities by facilitating the efficient access, processing, storage, and sharing of crucial digital data. The number of workshops and forums on this new computing paradigm should be increased, which will help the scientists, engineers and software developers start thinking about totally new scenarios where applications, simulations and experiments are closely coupled with large amounts of observational and empirical data, which would revolutionize science, not just in the new scenarios but in the way it will bring the computational, theoretical, and experimental scientists together who do not normally interact.

References:

[1] Climate Change Science Program, "Strategic Plan for the US Climate Change Science Program", *CCSP Report,* 2003.

[2] NSF, "Research Challenges in Distributed Computer Systems", *NSF Workshop Report*, 2005.

[3] NSF, "NSF's Cyberinfrastructure Vision for 21st Century Discovery", *NSF Cyberinfrastructure Council Report,* January 2006.

[4] DOE-Office of Science, "The Data Management Challenge", *Report from the DOE Office of Science Data-Management Workshops*, March-May 2004.

[5] Kosar, T. & Livny, N. "Stork: Making data placement a first class citizen in the Grid." In *Proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, March 2004.

Sustainable Software for Cyberinfrastructure

Reagan Moore[1], Arcot Rajasekar[1], Mike Wan[2], Wayne Schroeder[2]
Data Intensive Cyber Environments Group
{sekar, mwan, moore, schroeder @ diceresearch.org}
1University of North Carolina at Chapel Hill
2University of California, San Diego

We submit this position paper on software sustainability based on our nearly 15 years of experience in designing, developing, distributing and deploying cyberinfrastructure software that is being used across the world. The Data Intensive Cyber Environment (DICE - previously called the Data Intensive Computing Environment) Group has developed the Storage Resource Broker (SRB) and integrated Rule-Oriented Data Systems (iRODS) software. These data grids support data virtualization and data sharing across heterogeneous storage systems.

*** What kinds of software are most important to sustain and maintain in support of the NSF mission? What are the criteria that should be used to decide on priorities for sustainability and maintenance?**

Scientific research is based upon the ability to compare theory and simulations with experimental and observational data. The data are assembled in reference collections to enable comparison of future analyses with the current state-of-the-art understanding. The reference collections are published for use by the entire scientific discipline. This process is used by all science disciplines to document scientific progress and facilitate exchange of knowledge.

The scale of the science research questions has grown massively, both in terms of the number of researchers who collaborate, the cost of the equipment needed to conduct experiments, the cost of the sensors that take observations, and the size of the reference collections. Large projects minimize cost by promoting the rapid sharing and publication of research results to minimize the amount of effort invested in unproductive areas. Cost minimization examples include automation of sensor data rates by dynamically comparing current observations with prior observations. Data rates are only increased when significant events are detected. Reference collections are migrated to more cost effective hardware, and data administration tasks are automated. Current data scales are measured in petabytes and hundreds of millions of files. Projects (experiments, observatories, simulations) are growing in scale to collections with 100 petabytes of data.

Software environments that help minimize cost enable the investigation of a larger number of research questions, and have the highest priority for sustainability and maintenance. In particular, cyberinfrastructure software is needed that manages shared collections for the research teams, manages data distribution for large experiments, manages real-time sensor data streams for observatories, and builds digital libraries of simulations results. The cyberinfrastructure software organizes distributed data into collections, enables large-scale data analysis across distributed data, and enables longterm preservation of the reference collections. The technology that provides these capabilities is a data grid. Data grids form a key component of cyberinfrastructure.

*** What models for sustainability and maintenance of software exist, independent of federal funding, that can be applied effectively to the cyberinfrastructure software that supports NSF researchers and the NSF mission?**

We find that there are four important aspects to software sustainability:

1. Expanding the User base,

2. Extensibility and Adaptability of the software,

3. Organic software development with a strong user-feedback loop

4. Transition to a stable base such as a software foundation or industrial support.

User Base:

The importance of a sustainable user base cannot be over emphasized for a software system. Also in the case of open source software, an expanding user base is very much needed in the initial stages (first few years), and a sustained user base during the core development and "dig in" phase (next few years). The user base will need a stable software maintenance and support in the follow-on years after major development have been completed.

In a scientific research setting, as promoted by NSF, one of the best ways to make sure that software is adapted by an expanding user community is to provide tools or services that:

a. do not interfere with their research,

b. provide a much needed solution to a cyberinfrastructure problem

c. are easy to learn and use,

d. provide an expanding array of features that helps their scientific endeavor.

A system that is very easy to install, administer and use will lead to wider adoption by a broader audience. Any roadblocks at these levels will take away precious time and frustrate the scientists who will walk back to their suboptimal, but tried and true solutions. As scientists start using a new system, they will probably adopt only the minimal set of features that they need to solve a specific problem that they are facing. But as time goes on, the adoption of more involved features that help in advancing their research will become easier once they know and trust the system.

At these stages, to minimize frustration, one needs strong user support for multiple levels of expertise (not just at computer science levels). Financial support for such user liaison should be built into a system development process model.

Extensibility and Adaptability of software:

Any software that is widely applied will be extensible. One size never fits all. Building a system that can work from a small-scale to enterprise levels will be more useful. This will also allow for a scientist to start using the system at a small level and expand their usage among colleagues locally, and then globally.

Adaptation also has another aspect. Since problems are nuanced for each discipline, the software should be easily "changeable" to meet the needs of multiple disciplines. If this is not the case, one will be left with a software package that is the least common denominator and not useful to anyone in a significant manner.

Adaptability also means that a scientist can build their solution using the various building blocks that are provided by the software package. A lego-block type approach with service-orientation concepts is a very good paradigm for such adaptation. But the key should be that the integration should be easy to perform and intuitive to the end user.

Organic Software Development:

Normal software development follows the "cyclic" or "waterfall" approaches with separate stages for design and development. Usually in these approaches, the end user is brought in only at the very end. The end user is exposed to a "complete" product and an immense amount of time is spent in educating the client in the usage of the product. Moreover, since the design goals were set probably 4 or 5 years earlier, by the time the product is rolled out technological advancement may make some if not all parts obsolete,

and moreover integration with an advanced set of systems may need more time and adaptation than is possible.

We have found that this approach, though useful in an industrial setting, does not work well in a scientific arena. What is needed is an organic approach with the end user in the loop. We propose a design model where a small set of key features are developed in a short period of time (may be 6 months) and then thrown open to the scientists to use, test and give feedback. The system may not be complete, it might be a bit frustrating, but the fast turnaround feedback will be useful in the design and development of the next short cycle. The end user can make quick suggestions of what they want (probably new goals) and how thing can be improved and these can be plowed into the next short cycle. In this manner, the user is involved continuously in the large-scale design. The short-cycle organic design paradigm has the end-user involved continuously, can roll and adapt with the punches as technology changes under their feet and can become self-sustainable as it moves with a shifting goal.

Transition after maturity:

Any software that is deployed after a life-cycle of development and adaptation needs to be supported for a longer time for sustainable usage. In a scientific setting with open source software, such a sustainability model can come from either an industrial adaptation (with support for industrial strength software) or a software foundation - independent of the development team - where it can be maintained and sustained.

Moreover, this also relieves the designers from supporting the software for long periods of time. They can move on to developing cyberinfrastructure that can be a major extension to the current system – such as a complete redesign and paradigm shift - and be more productive. Financial support for such foundations needs to be built into the longterm goals of the system development. A large user base is a key to get such long-term support.

Our Experience:

The DICE team has more than 50 person-years of software development and deployment experience in cyberinfrastructure. In our approach to software development, we have adopted the above approaches. The SRB was built as a virtualization middleware for large-scale data sharing. Even before the word 'data grid' was coined several releases of the software were issued for adoption in the wide-spread NPACI partnership. Key features of software sustainability built into SRB includes:

- short organic development cycles with user-feedback based design changes

- modular design that made changes very easy for the developers

- very easy to install and administer. The installation can be done in minutes and we had one SRB administrator take care of 15 data grids!

- very easy to use. We provided tools that are intuitive and known: Unix commands that have a familiar structure, SRB Explorer that is similar to Windows Explorer), a web-based tool for ease of use and a set of useful Java classes for easily building domain-specific portals.

- integration with a large number of peer software to extend the usability of SRB. SRB was able to integrate with nearly 30 different types of software systems by the time its major development cycle ended.

- being vendor agnostic and platform-independent. Even though the system was developed using the C language (chosen because of the speed needed) the system was coded so that plug and play of any number of vendor product becomes easy (we saw integration times in terms of days)

- user support - our whole team was geared to helping the user. Our rule-of-thethumb was that the user gets a reply within a few hours and a solution within a day or so unless it is new feature development. We worked closely with the users to debug as well as design new features.

- wide use of software engineering tools - CVS, continuous build testing, NMItestbed validation, mailing lists, bugzilla and wiki.

SRB is still being used and supports more than 3 Petabytes of data all over the world.

One of the main outcomes of several years of SRB development is that we saw a need for a major paradigm shift. The user requests for new features became overwhelming even for a modular system such as SRB. Hence, we wanted to build a new system that is more adaptive and changeable (preferably by the users themselves). The outcome of this is the NSF supported iRODS middleware system.

In iRODS, we coupled a client-server peer-to-peer system that has data and trust virtualization as in SRB with policy-virtualization using a server-side rule engine. In iRODS, a user can form rules that can be fired based on triggers and perform complex workflows at the data site. The building blocks for these rules are called micro-services which have well-defined interfaces and can be easily chained together to form rules and workflows. An intriguing aspect of the iRODS rule systems is its ability to "clean up" on failures. This is possible because the rules themselves have a recovery section that is defined when the rules are defined and executes on failure. This helps in not only trying alternate rules but also makes sure that the system is not left in an unstable state.

In iRODS, we have adopted the same organic short-cycle design. Through user feedback, design goals can be changed even when following a very broad strategy. The user base in iRODS is not passive. Because of the ease of development of rules and micro-services, the users themselves shape and adapt their iRODS system to fit their needs. Many of these rule sets as well as micro-services are becoming part of our releases, after an initial test and review. The mantra of ease of use is also part of the iRODS development, as we provide familiar tools and extension for accessing data.

Adaptation of some useful interfaces such as Fedora, Dspace, PAWN and such are making the system usable by a wide audience.

We have also formed a non-profit foundation for managing long-term viability of the iRODS independent of the development team. We hope to attract other developers and users to this foundation and take charge of sustainability of the software system.

*** What role should the NSF and other funding agencies have in sustaining funding for important software? How can federal agencies best coordinate and achieve efficiencies of scale?**

Agencies should provide funding for core development and for applications of the software across scientific domains. Support is also needed for maintenance, hardening, testing and documentation after a core development period. Further support should be through community buy-in and from usage in large-scale funded and self-funded projects.

*** Are there dangers that heavy support of one particular open source system will stifle diversity of research? How should one choose software to sustain? How should one scale back when a particular system no longer seems promising but still has significant use? How does this support model compare to commercial systems where support is not guaranteed? Do open source systems require more or less operational support than commercial products?**

Funding should be diversified to ensure multiple approaches are explored. However, no system should be supported that is not capable of interoperability with other solutions. If it is possible to migrate collections between solutions, then the best solution (most cost effective and having the required features) will be self-selected by the user community. Software must be widely used if it is to become generic infrastructure, Within the academic community, open source software is strongly preferred because it can be modified to meet project specific requirements, is freely available, and is able to rapidly evolve to incorporate new methodologies. However, open source software is a doubleedged sword. The maintenance and support costs require the development of local expertise in running the software. This is a labor cost, but it is ameliorated by the transfer of expertise from the developer to each institution that uses the software. Open source software is a knowledge transfer mechanism that promotes enrichment of expertise at participating sites. Thus it is widely used in academia.

*** How can new strategies for sustainability of open source and community source software be employed to help advance NSF goals?**

Research now is conducted at an international scale. Software systems are needed that are used in support of international research collaborations. There is a corresponding commitment to international development collaborations. Data collections now are assembled on a global basis, with observation sources scattered around the world, researchers distributed across multiple continents, and software developers distributed across many countries. In particular, data grids are deployed as national infrastructure that tie together academic and research institutions within a nation. The development expertise to meet the wide range of demands resides in the participating nations.

The iRODS data grid contains contributions from developers in Europe, the US, the Far East, and Australia. These include clients for accessing the data grid, security extensions to improve interactions with grid software, micro-services for data manipulation, and structured information resource drivers for specific data formats. Sustainability becomes easier when the software becomes a de facto standard used across multiple communities and nations.

Sustainable Software Ideas for the Next Generation Research Grid: Standards, Interoperability and Software Development

David E. Hudak, Ph.D., Tom Bitterman, Ph.D., Neil Ludban, Stanley C. Ahalt, Ph.D.
Ohio Supercomputer Center
Columbus, OH
{dhudak, tbitter, nludban, ahalt}@osc.edu

The Next Generation Research Grid (NGRG) [Killeen2008] will require maintenance (and obsolescence) of existing software as well as development of software to bring new capabilities. For example, the NGRG vision calls for "pathways for the integration of a wide range of cyberinfrastructure (CI) resources and new providers". Achieving this goal necessitates an increase in the number of applications, grid services and resource providers (RPs). As a consequence, grid software infrastructure and NGRG policies will have to support an increasingly diverse set of services and service lifecycle models (such as those required by throughput computing or for interactive access).

In our experience, writing reusable software is more than making source code available; reusability must be constantly considered during software development. Similarly, sustainable software is more than software maintenance (or "open source"); sustainability comes from active communities of users, developers and experts both external and internal to the software development process. We believe that the NSF should foster the creation of sustainable grid infrastructure software by focusing on the following areas (1) grid infrastructure standards, (2) interoperability testing, (3) software incubation and (4) software product development.

Standards are defined for various layers of the data communication protocol stack (IEEE 802 standards for ethernet, IETF standards for IP, W3C standards for web services, the Globus Alliance and the Open Grid Forum for grid services). Support for standards being developed in the Open Grid Forum should be a major emphasis for NSF. This support could come in a variety of ways such as requiring OGF standard compliance (or development) as part of NGRG services or programs targeted at standards generation.

Recommendation: The NSF should provide ongoing funding for a diverse CI standards team focused on developing and promulgating standards for the emerging cyberinfrastructure. These standards should cover national, regional and campus cyberinfrastructure.

In order to create sustainable software, the TeraGrid Planning Process identified a need for "dedicated testbeds" [Killeen2008] for NGRG services. A natural analogy comes from the wireless LAN industry, in which there is a standards body (IEEE 802.11) and an industry consortium (the WiFi Alliance). Despite the efforts of their authors, many standards contain some ambiguity. Implementers make assumptions when faced with these ambiguities. In these cases, testing is necessary to ensure that multiple implementations of a specification interact correctly. Interoperability tests have been discussed and published, but there exists no standing organization to develop comprehensive interoperability tests and certify testing results. This is a critical area as the grid computing space expands with standards such as those proposed as a logical next step for the commercial "cloud computing" space [Armbrust2009].

Recommendation: The NSF should provide funding for a CI interoperability team. This interoperability team should be charged with cooperatively generating a suite of tests, acceptable test outcomes, and metrics that can be used for standard certification covering national, regional and campus cyberinfrastructure.

We have found it useful to categorize software development activities as either software incubation or software product development. Informally, we use "software incubation" to describe time-limited software development projects of modest scale. In typical software incubation projects, the application's author is its primary user, the application need only run for a limited amount of time on a limited number of systems. Similarly, we use "software product development" to describe products that are to be ongoing, widely distributed, larger projects. These projects must address many other factors including testing, error handling, performance, scaling, standards conformance, multi-platform support, documentation, tutorials, and installation.

Recommendation: The NSF should permit funding for software incubation, development, and support to be included in future CI proposals, in particular those proposals that are directed at the development of community-oriented CI products such as, but not limited to, innovative parallel libraries, domain-specific grid "stacks", storage management, collaboration tools, visualization (including remote visualization), and portal components.

Many publicly visible software development projects are carried out in the open source (or community source) communities, e.g., Linux, Apache, Python and Eclipse. These projects have developed similar online collaboration/feedback mechanisms (e.g., source control repository, mailing lists, bug tracking, wiki, chat) functions bundled in applications like SourceForge. However, technology alone is not enough to create a successful open source project. In addition to the technical infrastructure, successful open source projects [Fogel2007] exhibit a number of similar traits, include (1) intersection of personal and professional interests for developers, (2) passionate or charismatic lead developers (such as Linus Torvalds or Guido van Rossum), (3) low barrier to entry (simple, established mechanisms for project installation, collaboration and contribution) and (4) active user and developer communities. If NSF wants to see such communities develop around critical NGRG capabilities, it must establish funding and evaluation mechanisms for all aspects of software development.

Recommendation: The NSF should establish evaluation criteria and funding mechanisms that support software development, release, and life-cycle improvement. This is particularly critical for relatively lower-use software that is essential to the nation's escience objectives but which may not initially have a broad user-base or immediate commercial potential. Funding should be provided to support software development technologies including repositories, user mailing lists, bug-tracking, and testing. Further, NSF should consider developing mechanisms to allow intellectual property value to accrue to some software development activities outside of the open-source community.

References

[Armbrust2009] Michael Armbrust, et. al., "Above the Clouds: A Berkeley View of Cloud Computing", 2009. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

[Fogel2007] Karl Fogel, "Producing Open Source Software: How to Run a Successful Free Software Project", 2007. http://producingoss.com/

[Killeen2008] Timothy L. Killeen, et. al., "The Next Generation Research Grid: A Path Forward Final Report", 2008. http://www.teragridfuture.org/system/files/NGRG_Report_Final_0.pdf

Sustaining Capabilities Not Codes By Architecting for Innovation

James D. Myers, Robert E. McGrath
National Center for Supercomputing Applications (NCSA)
University of Illinois, Urbana-Champaign

1. Introduction

It is important in considering sustainability to ask what needs to be sustained. Specifically, is it software products, or the capabilities enabled by that software? In this position paper, we argue that sustaining capabilities and architecting to enable change are often the better choice, particularly when one considers the costs and consequences of maintaining a specific software product. Further, these approaches are not independent – software products architected to support change are themselves more maintainable.

There are a wide range of technical and social issues that influence the relative merit of these alternatives. In our experience as part of the community developing scientific cyberinfrastructure, it is very rare to see specialized software products re-used across projects. Though there are exceptions, even successful products are often more successful in spawning new variations and competing products than they are in becoming ubiquitous themselves:

- Later products can incorporate components that were not available to the original developers and can add/evolve general capabilities to better support specific communities,
- Variants may integrate better with other communities' existing infrastructure than the initial product,
- Communities can consider the combination of CI developers familiar with their domain and the availability of support as arguments in favor of competing products, and
- Funding for maintenance is often more readily available as part of developing more advanced capabilities.

While all of these issues can be addressed with additional funding, which may be a necessary part of the solution, it is also worth noting that these issues are not as relevant if the goal is to maintain capabilities rather than products and if software is designed to support migration to new products over time. Consider the case of NCSA Mosaic and httpd in the early days of the WWW. Clearly investment by NSF and other agencies was critical to create these initial free implementations of the Web protocols. However, the success of the Web has been driven by the Web's open HTTP and HTML standards and the availability of multiple browser and server implementations rather than the continuing viability of the initial implementations. Once the idea of the web caught on, support for Mosaic per se was not so critical.

While the Web can be dismissed as unique, there are numerous other examples where standard interfaces and protocols, particularly those that provide end-user extensibility, have lead to long-term sustainability of capabilities. In the following sections, we explore the idea of sustaining capabilities in more detail, discuss its application to scientific cyberinfrastructure, and conclude with a discussion of how this reframes the discussion of sustainability and leads to different approaches to achieving long-term sustainability of scientific cyberinfrastructure.

2. Architecting for Innovation

HTTP and HTML are two examples of a class of standards that cleanly separate how something is done from what is being done. HTTP defines how one can GET and PUT blocks of content across the Internet but does not constrain what they are or how they are generated. HTML specifies page formatting but not what you can discuss on the page. XML similarly standardizes syntax but does not constrain content, and service-oriented architectures have an analogy with HTTP (quite direct for RESTful services). The Internet itself is designed this way, with TCP/IP specifying how to route packets not what they can contain. Other examples of this approach in today's CI include the Pluggable Authentication Module API [4] and Java Authentication and Authorization Service (JAAS) (http://java.sun.com/products/jaas/). These abstract APIs insulate applications and services from both the specific authentication technology deployed and the administrative policies at various locations. They define the handshakes required between users, applications, and authentication services required to perform authentication but keep details of how authentication is performed out of applications. This enables applications to be reused in different security environments and for communities of users to upgrade to new authentication services over time without involvement of application developers. An application could be deployed using usernames and passwords at one site and Grid credentials at another (or at a later date) with no changes to the application software itself.

All of these standards encourage creativity "at the edges," and they have supported continuing evolution and expansion of capabilities over years and, for some, over decades. For the web, the separation of the means to reference, link, and format resources from the content being conveyed was the critical design principle that led to reusable browsers and servers and empowered web users and developers to innovate without the need for, or the expectation of, central coordination. The challenge in developing sustainable scientific CI can be framed in similar terms: designs should separate scientific concerns and operational concerns from the base capabilities and capability interactions required.

3. Designing Scientific Cyberinfrastructure for Innovation and Sustainability

Clearly many of the approaches outlined above have been adopted in scientific cyberinfrastructure projects with very positive results. To a significant degree, it has become possible to assemble off-the-shelf components for web interfaces (e.g. portals and wikis), web services, and collaboration tools to support scientific communities. While this has been a significant step forward, it has not eliminated the difficulties in sustaining systems. As has been discovered by many NSF-funded cyberinfrastructure projects, choices made in terms of security, data and metadata storage, and computational processing model often add dependencies that make it very difficult to integrate software across projects or to incrementally advance infrastructure despite the similarities in, for example, the use of web services and portal standards.

We have argued elsewhere that these issues are not fundamental limits on how much interoperability can be achieved but are instead indications of where additional extensible standards are needed [2, 3]. They also suggest that, due to the nature of scientific work and the distributed, non-hierarchical, overlapping nature of scientific communities, that the scientific cyberinfrastructure community is tackling issues that have not yet been addressed by the larger business-oriented community. Specifically, we have argued elsewhere that standardization of workflow representation, provenance, and data andmetadata representation a) are possible if done in the extensible style of the standards outlined above and b) would, in combination with broader use of standards such as JAAS in security, significantly increase software reusability and the ability to maintain capabilities while upgrading individual software components over time.

A critical point in these arguments is that it is a false dichotomy to think that one can only standardize either the lowest common denominator of functionality or a superset of capabilities that can cover all requirements. Clearly standards such as those listed above do neither, e.g. XML does not specify a standard way to represent chemical or mathematical information, nor does it prevent development of such standards; it simply standardizes a syntax and consequently software tools and interfaces, for managing hierarchical information. In the same way, efforts such as the Open Provenance Model (OPM) initiative are standardizing just the core concepts of causal chains of data 'artifacts' and processes, leading to increased interoperability of provenance tracking systems without limiting how granular or detail provenance information can be [1]. We see the combination of semantic web and content management standards the same way; they provide a common abstraction for coordinated management of data and metadata in open distributed environments without constraining what metadata or data can be handled or specifying details of replication/caching/data distribution strategies. Software such as NCSA's

Tupelo (http://tupeloproject.org), iRODS (http:/www.irods.org), and the semantic wrappers used in bioinformatics are all variations on this theme.

4. Sustaining Cyberinfrastructure

One of the drivers towards these approaches that we have seen are NSF's large, operations-oriented projects, particularly the proposed environmental observatories (OOI, NEON, WATERS). These projects have had significant community input and explicitly address the need to build an infrastructure that can be sustained, and sustained costeffectively, for 30 years or more. Invariably, these discussions have lead to approaches that incorporate 'loosely-coupled', 'innovation-at-the-edges', 'scalable-not-scaled' designs that incorporate ideas such as those above as well as virtual machine infrastructures, 'enterprise bus'-style event processing, and other elements. Further, many of these projects have seen a need to coordinate across projects, both to support scientific inquiry across projects and to create a standards-based market that would allow additional aspects of their infrastructure to be commoditized. In particular, there has been consensus at workshops including the "Cyberinfrastructure for Environmental Observations, Analysis, and Forecasting: A Cyberinformatics Forum" (http://www.cyberobservatories.net/events/workshops/20080505/) held last spring recommending the creation of a Federation of Environmental Observatory Networks (FEON) as a coordinating body that could pursue standardization and advocate for the type of extensible standards discussed here.

We believe that, with the coinage of the term cyberinfrastructure and the pursuit of longterm efforts such as NEES, the environmental observatories, DataNet, and others, NSF is shifting into an era where lifecycle costs of software capabilities, rather than just development costs or the return on investment calculated only from the reuse of a particular software product, become a dominant concern. This in turn argues for increased support of activities to support interoperability, for development of reference implementations rather than one-size-fits-all solutions, and, most critically, additional means such as FEON for operations-oriented efforts to have input on decisions concerning support and hardening of software. Such an overall approach might, as discussed in the FEON workshops, lead to a standard for data and metadata backup that could become a means for projects to outsource their back-up data stores to each other or to commercial providers as well as an export mechanism for data from multiple projects to be integrated by third parties to support cross-disciplinary research. Workflow and provenance standards could likewise increase interoperability incrementally and enable projects to migrate to newer research, open source, or commercial workflow systems over time.

As noted earlier in this position paper, the approach we're arguing for is not at odds with a call for increased support for sustaining software products. More attention and funding is needed to support sustainability. However, too much emphasis on maintaining specific projects and trying to encourage/enforce their reuse is essentially an effort to create an artificial monopoly which can then lead to well known consequences; while monopolies can create initial efficiencies, they can lead to stagnation and higher long-term costs as well. A balanced approach providing additional support for maintaining software that also supports the development of extensible standards and input from operations-oriented projects on where commoditization is appropriate stands the best chance of maximizing the long-term return on NSF's cyberinfrastructure investments.

References

[1] Moreau, L., J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson, The Open Provenance Model: An Overview, in Provenance and Annotation of Data and Processes. (2008) 323-326.

[2] Myers, J.D., J. Futrelle, J. Gaynor, J. Plutchak, P. Bajcsy, J. Kastner, K. Kotwani, J.S. Lee, L. Marini, R. Kooper, R.E. McGrath, T. McLaren, A. Rodriguez, and Y. Liu, Embedding Data within Knowledge Spaces, (2009), http://arxiv.org/abs/0902.0744v1.

[3] Myers, J.D. and R.E. McGrath. Cyberenvironments: Adaptive Middleware for Scientific Cyberinfrastructure. In: Adaptive and Reflective Middleware. (2007).

[4] Samar, V. and C. Lai. Making Login Services Independent of Authentication Technologies, SunSoft, Inc, 1996, http://www.sun.com/software/solaris/pam/pam.external.pdf.

Sustaining Software for Long-Lived Science Stakeholders of the Open Science Grid

Ruth Pordes for the Open Science Grid Consortium, ruth@fnal.gov, March 2009

The Open Science Grid Consortium provides a multi-disciplinary collaboration in support of the computing needs of long-lived physics, as well as other domain science, communities across DOE and NSF. The software systems are developed through collaboration across computer science and information technology groups with the internationally scoped science communities. For the large physics experiments, CS and IT provide common and reusable capabilities and the communities themselves provide domain specific software closest to the user applications. Clearly the computing and software environments evolve significantly in functionality and characteristics over the lifetimes of their use. Within the Consortium, the user communities depend on and share technologies through the OSG's Virtual Data Toolkit distribution and support (a joint team at the University of Wisconsin Madison and Fermilab). They have access to a common build and test environment Metronone (from the University of Wisconsin Madison) and use the OSG Integration Grid for integration and testing. There is heavy reliance on open source technologies such as Condor as well as general toolkits such as Apache/Tomcat and databases such as mysql. The needs and experiences of our stakeholders in this environment drive our recommendations below:

How long is software sustained and how should this be prioritized?

This is best to be user community driven. When there are scientists who are interested in the data, priority is attached within the community to maintain the mechanisms to use it. "Scientific interest" includes the broad range of science, historical record and educational study, as well as the interest in new data with increased depth, breadth and accuracy.

What should be the role of government agencies?

Despite large scale and long-lived investments in software development and research we have not yet achieved the goals of universal, usable, robust, secure, software frameworks and toolkits for scientific computing. It is ever more important to have a coordinated approach across the sponsoring agencies and across international boundaries. Cross-agency and cross-government involvement is essential to address: definition and agreement on metrics and processes for validation and auditing of the needs and priorities; a comprehensive community-based vision, architecture and prioritization for software capability and (r) evolution; attention to re-use and evolution of existing codes; and reduction in inefficiencies of multiple-implementations of the same thing.

What programs and studies are needed to provide the knowledge, research and implementations?

As the amount of software increases with time the long-term support, management of maturity, and evolution of the codes becomes ever more important. Attention is needed to avoid duplication of effort. A coherent strategy is required in order to hope to cover the complete set of needs. A coordinated approach to cost-effective software development becomes crucial. In depth thought and knowledge are needed to augment the "develop it and they can use it" approach. In summary:

• Collection of requirements and prioritization of common needs across all communities.

• Systematic assessment of software development risk and maturity management.

• Auditing of design and code quality and security.

• Implementation of validation models processes and tools.

• Understanding of and response to of software aging.

• Planning for software end of life.

A significant investment in human training and needed expertise needed must also be sustained in parallel with the software. Scientists emphasize that support from technical software and IT experts are crucial to their ability to successfully leverage computational methods for their research.

The Importance of Long-Term Science and Engineering Infrastructure for Digital Discovery

Nancy Wilkins-Diehr[1], Jay Alameda[2], Kelvin Droegemeier[3], Dennis Gannon[4], Stuart Martin[5]

1. San Diego Supercomputer Center, University of California at San Diego
2. National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
3. University of Oklahoma
4. Microsoft
5. Argonne National Laboratory

wilkinsn@sdsc.edu, jalameda@ncsa.uiuc.edu, kkd@ou.edu, Dennis.Gannon@microsoft.com, smartin@mcs.anl.gov

This is an abridged version of a white paper submitted to the National Science Foundation Office of Cyberinfrastructure, September, 2008.

1. Introduction

In the 17 years since the development of Mosaic, the world has undergone a tremendous transformation. The internet is a fundamental component of modern life. Millions rely on it for travel arrangements, banking, news, shopping, and much more. The internet has had a tremendous impact on the conduct of science as well – an impact we can feel today and one which will surely be felt even more in the coming decades.

In order for scientists to benefit from the developments of the information age, they must have infrastructure they can rely on to assemble the tools necessary to perform research in today's environment. This infrastructure includes not only a growing number of individual components like high performance computers, high speed networks, sensor networks and databases, but also an infrastructure overlay which assembles these components so that scientists can use them productively.

Many scientists today develop their own infrastructure, assembling the components they need. The NSF's Information Technology Research (ITR) program is one recent program where scientists apply technology and assemble the components of cyberinfrastructure to address the most challenging domain science questions.

The TeraGrid Science Gateways program works with many such groups who are interested in incorporating high-end compute and data resources into their own infrastructure. Our work has uncovered similarities in approach to building infrastructure amongst developers from widely varying domains. We have seen significant increases in research productivity through the use of gateways, but have also seen limitations to that productivity without a long-term funding model for successful gateways and the infrastructure they provide.

We believe a single NSF-wide program to develop coordinated, sustainable infrastructure is critical in this area. Without it, we will not see the sustained increase in research productivity possible through the use of gateways.

2. Problem Description

There have been a number of NSF reports citing the need for improved cyberinfrastructure – both to increase scientific productivity and to enable new techniques and discoveries. These reports are listed at the end of this document. A full discussion of the recommendations and their relationship to the development of science gateways is included in the unabridged version of this paper [1].

Through work in the TeraGrid Science Gateways project, we have had the opportunity to work with a number of groups and have observed many similarities in underlying tasks. For example most gateways must develop solutions for data management, authentication, execution management, accounting, user workspaces, collaboration tools, visualization and usage statistics.

As these projects develop in isolation there can be quite a bit of duplication of effort. For example there are similarities between accessing radar data and incorporating it into workflows that include simulations on supercomputers and accessing seismogram data and doing the same. Developers enabling access to digital data returned from a telescope have much in common with developers providing access to digital data from an electron microscope. Those providing access to data from the sea floor have much in common with those providing access to data from Antarctic. Without a common infrastructure, development of unique tools and techniques that can provide results for the near-term result.. These often do not lead to growth or stability for advancing the building blocks of cyberinfrastructure.

3. Benefits of a Coordinated, Sustainable Gateway and Infrastructure Program

We think there is great potential in the development of an agency-wide science gateways program to provide fundamental next generation capabilities for the nation's scientists. A coordinated, far-reaching program can: 1) reduce duplication of effort, 2) increase continuity of development, 3) contribute to a diverse workforce, 4) enable scientists to study complex multi-disciplinary problems, and 5) meet NSF's strategic goals of discovery, learning, research infrastructure, and stewardship.

3.1 Increased scientific productivity and enhanced problem-solving opportunities

Science gateways and the infrastructure on which they rely will allow scientists to study more complex problems productively. Transformative science today is often not easy to accomplish by isolated research groups in a single laboratory or office. In many fields, advanced tools and environments are necessary for the most pioneering work. Talented gateway developers who can create and assemble the right tools and environments will free scientists to focus on challenging interdisciplinary scientific problems, assemble the best research teams regardless of location and provide mechanisms for others beyond the research team to bring new perspective. Gateways make it possible to conduct leading edge research, using the most advanced tools and assembling the most capable teams to address the most challenging problems. Decoupling gateway development from and middleware development provides scientists with a certain level of continuity while allowing developers to take advantage of new hardware and software infrastructure developments.

3.2 Lowered costs through less duplicated effort

A coordinated, long-term science gateway and infrastructure program would reduce duplication of effort we witness today. We recommend a 10-year program to develop, deploy and operate persistent science gateways with a review at 5 years. Such a program must also shift focus from research explorations to infrastructure deployment and the incorporation of robust system engineering practices this implies. While persistent gateways must be designed to facilitate research on the frontier and truly meet the needs of their defined user communities, the act of building the gateway itself should not be research but instead a true infrastructure deployment project.

Sustained funding must include a very important determination of which projects to fund. We recommend that a sustainable program be initiated through a detailed cross-directorate and perhaps multi-agency study. Such a study might begin with directorate and agency workshops to understand the most pressing needs in different domains and identify common infrastructure needs. Biologists view the Protein Data Bank as indispensible to their work. Workshops would help uncover similar fundamental needs in other domain areas. Some needed infrastructure may consist of curated data collections. Others might also involve simulation, visualization and analysis capabilities. Some might provide collaboration tools or workspaces that allow scientists to store and share results. Still others might allow researchers to generate and store complex workflows or access instruments, sensor or radar data that have limited exposure today.

Merit review and regular evaluation and assessment are critical in any persistent program. Criteria to evaluate success must be factored into a long-term gateway and infrastructure program.

3.3 Increased workforce development

The nation needs to make use of an evolving, diverse workforce to maintain global leadership in science and engineering. Diversity here refers both to the makeup of the workforce as well as skill sets of the workforce. Development of science gateways address both of these. There is considerable interest in the use of gateways by educators, particularly those in remote locations and those who are not located at traditional research institutions. Students and educators at these sites can gain access to cutting edge cyberinfrastructure through science gateways. In addition, gateway developers themselves have highly valuable cross-disciplinary skills. As scientists rely more heavily on gateways there will be an increased need for this type of skill set.

3.4 Improved public perception of the value of science

Because of their online availability, gateways can provide access to cutting edge cyberinfrastructure and domain specific educational tools. A November, 2006 study by the Pew Internet and American Life Project entitled “The Internet as a Resource for News and Information about Science” found that 87% of internet users use the resource for research, half of all internet users have been to a site that specializes in scientific content and that those who use the internet to gain scientific information are more likely to believe that scientific pursuits have a positive impact on society [2]. The existence of gateways with information to fit a variety of interest levels can improve public perception of the value of funding science.

4.0 Conclusions

Sustainable science gateways have many benefits, both for scientists and the population at large. Gateways, however, must rely on a robust infrastructure to be successful. This infrastructure includes airtight software - for workflow development, visualization, grid computing, etc. Without a foundation of reliable infrastructure, it is impossible to develop reliable gateways. Without reliability, gateways will never be seen as indispensible tools that scientists use to increase productivity and focus on the most challenging science problems. Software sustainability is extremely important to a successful science gateway program.

References

[1] Wilkins-Diehr, N., Alameda, J., Droegemeier, K., Gannon, D., “The Importance of Long-Term Science and Engineering Infrastructure for Digital Discovery”, submitted to the NSF’s Office of Cyberinfrastructure, September, 2008.

[2] Horrigan, J. “The Internet as a Resource for News and Information about Science”, Pew Internet and American Life Project, November 20, 2006, http://www.pewinternet.org/pdfs/PIP_Exploratorium_Science.pdf.

Background Materials

[1] Atkins, D., et al. Revolutionizing Science and Engineering through Cyberinfrastructure: Report of the National Science Foundation Blue-Ribbon Panel on Cyberinfrastructure. National Science Foundation, Washington, DC, USA, 2003.

[2] Scalise, G., Reed, D., et al. Leadership Under Challenge: Information Technology R&D in a Competitive World, President’s Council of Advisors on Science and Technology, August 2007.

[3] National Science Foundation Strategic Plan, FY2003-2008, September 30, 2003.

[4] Zimmerman, A., Finholt, T. Growing an Infrastructure: The Role of Gateway Organizations in Cultivating New Communities of Users. In Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP ’07) (Sanibel Island, Florida, November 4-7, 2007), ACM Press, New York, NY, 2007.

[5] Science and Engineering Indicators 2006, National Science Foundation report, February 2006.

[6] Cyberinfrastructure Vision for 21st Century Discovery, Report of the National Science Foundation’s Cyberinfrastructure Council (CIC), March, 2007.

[7] Computational Science: Ensuring America's Competitiveness, Report of the Computational Science Subcommittee, President's Information Technology Advisory Committee (PITAC), June 2005.

[8] Berman, H., Bourne, P., Westbrook, J. The Protein Data Bank: A Case Study in Management of Community Data, Current Proteomics, 2004, 1, 49-57.

[9] Bement, A. "Testimony Before the Senate Commerce, Science & Transportation Subcommittee on Technology, Innovation, and Competitiveness", April 19, 2007. http://www.nsf.gov/about/congress/110/alb_basicresearch_041907.jsp

The iPlant Collaborative Position Paper
Submitted to: Indiana University
Re: Cyberinfrastructure Software Sustainability and Reusability Workshop

The iPlant Collaborative requests consideration to participate in the NSF-sponsored Cyberinfrastructure Software Sustainability and Reusability Workshop to be held March 25 – 27, 2009, in Indianapolis at Indiana University.

What is the iPlant Collaborative?

The iPlant Collaborative (iPlant) was funded in February 2008 by NSF as a project to "create a new type of organization – a cyberinfrastructure collaborative for the plant sciences" – that seeks to transform the way plant biologists answer Grand Challenge questions and collaborate in the data-laden and cross-disciplinary research environment in which we now live.

Grand Challenges in the plant sciences are research questions that are currently intractable with conventional approaches. For example: What are the genetics of species range limits? How do we improve crop yield under environmental stress? Which genes and pathways have an effect on ecophysiological behavior of plants? The iPlant Collaborative focuses on using cyberinfrastructure development as one way to resolve such questions.

The identification of grand challenge questions is a community-driven effort. Of the nine workshops proposed to the Board of Directors, five were supported and held at Biosphere 2 in Arizona between October and December 2008. These community-driven workshops were designed to identify specific grand challenge problems where cyberinfrastructure could be designed, developed, and applied to significantly advance our knowledge of plant biology. The topical focuses of the workshops included the evolution and development of plants, the tree of life, modeling the growth of plants, the adaptation of plants to their environment, and associating plant genotypes with phenotypes. In total, iPlant hosted nearly 250 plant scientists, computer scientists, educators, and industry leaders representing over 130 institutions, corporations, and interest groups in the five workshops. Another 50+ community members participated in the workshops remotely.

iPlant also sponsored a brainstorming workshop in January 2009 on the cyberinfrastructure required to address Grand Challenges in the plant sciences. More than 40 cyberinfrastructure community experts described their experiences building cyberinfrastructure and advised iPlant on lessons learned and pitfalls to avoid. The issue of sustainability was a cogent and frequently mentioned topic during the workshop, as numerous participants representing both grant-funded projects as well as commercial and research enterprises were also grappling with how to meet this challenge.

Self-forming Grand Challenge Teams from the community are now working on collaborative project proposals to develop Discovery Environments (DE)--the cyberinfrastructure needed to address and solve the team's grand challenge. Ultimately, iPlant's goal is to address the cyberinfrastructure needs of different grand challenges ranging from the molecular, cellular, and developmental to the organismic, ecological, and evolutionary plant sciences.

At its core, iPlant is a community-building and educational enterprise. Grand Challenge Teams and iPlant core staff will work together, using the cyberinfrastructure we are building, to educate the next generation of plant scientists. The iPlant Collaborative's core staff is located at the University of Arizona, Cold Spring Harbor Laboratory, University of North Carolina at Wilmington, Arizona State University, and Purdue University.

The iPlant Collaborative's Cyberinfrastructure Development Goals

The cyberinfrastructure (CI) developed by iPlant will provide the community with two main capabilities to enhance research and education: 1) access to world-class physical infrastructure, such as persistent storage and computer power via local and national resources, and a platform that promotes interaction, communication, and collaboration in the community and that advances the understanding and use of computational thinking in plant biology.

The cyberinfrastructure framework consists of a comprehensive combination of hardware (campus cluster and TeraGrid-based resources, primary and secondary data repositories, and advanced visualization facilities), software (open source, developed by a distributed team of developers and programmers), network fabric, and a multidisciplinary team of experts. Distinct Discovery Environments (DE) will consist of community collaboration spaces, novel mathematical and computational approaches, semantic data analysis/ discovery tools, an underlying cyberinfrastructure for access, analysis, and collaboration, and both feedback processes and social network analyses for studying, evolving, and refining the DEs.

The iPlant Collaborative's CI Development Process The group currently developing the cyberinfrastructure for iPlant consists of three sub-teams: CIT (Core Infrastructure), EA (Enterprise Architecture) and IST (Integrated Solutions). CIT is responsible for the basic infrastructure for the collaborative, which includes compute and storage platforms, systems software, relational databases, virtualization and authentication/ authorization. The EA team is establishing a foundational environment for application development and support, including software for collaboration, scientific workflows, semantic data integration, metadata management, name services, vocabularies, ontologies, data/text mining, grid and web service middleware. IST is responsible for assisting in the design and development of Discovery Environments. Each DE is a software platform custom-designed to help biologists in the community address and solve a Grand Challenge (GC) problem. It provides both a virtual meeting place for a GC team, and it allows the team to access the underlying physical infrastructure. In particular, a DE allows GC team participants to access relevant data sets, integrate across them to identify connections, visualize them in ways that allow the 'big picture' to appear, manipulate the data with analytic tools, and share results by facilitating computational steering.

The Sustainability Challenge for the iPlant Collaborative's CI

At the conclusion of the iPlant Collaborative project, it is envisioned that huge repositories of both software and data will be available for the plant sciences community. To properly ensure these systems continue to be used after iPlant ends and to possibly engage third parties to continue the effort, the following policies have been enacted:

Components: As much as possible, all software developed will consist of components, which at most levels, will allow the insertion or deletion of systems into the architecture. This standard applies to all levels of the iPlant internal software architecture, from middleware to the use of the Model-View-Controller (MVC) paradigm for all Discovery Environment work.

Engagement: The CS and CISE community will be actively engaged and encouraged to participate and/ or collaborate with the iPlant scientific and development teams.

Education and Outreach: EOT will be a key and integral component of all CI developed by iPlant. Training the next generation of plant scientists in the use of iPlant tools should foster improved community adoption of the tools.

Life Cycle Management: We will establish a working group composed of 20-25 volunteers from thecommunity. This group will be tasked with developing policies and mechanisms for data and software retention, archiving, and retirement. All of these issues fall under the general topic of information life cycle management (ILM).

Software: To encourage the widest distribution of the software iPlant produces, we will distribute all novel software under the Academic Free License version 3.0 (http://www.opensource.org/licenses/afl-3.0.php), which allows software to be reused, modified, and redistributed provided that attribution of the original work is maintained. Much of our software will depend on third-party components, however, and other licenses may apply to these dependencies. We will make every effort to use open source, third-party components whenever technically feasible, and will provide the community with visible alerts in cases in which we have been unsuccessful in doing so.

Integrated Data: All datasets that we produce by a process of integration of third-party data will be distributed under the Creative Commons 3.0 “by” license (http://creativecommons.org/licenses/by/3.0/), which allows the data to be copied, redistributed, and adapted provided that attribution of the original dataset is maintained. In some cases, additional third-party restrictions will apply to the dataset. Whenever possible, we will avoid using restricted datasets due to the complexity of redistributing derivative works, but when unavoidable, we will alert the community to these restrictions.

Novel Data: Datasets derived by grand challenge teams will be released under terms of the “Fort Lauderdale data sharing agreement” (http://www.wellcome.ac.uk/assets/wtd003207.pdf) in which data producers release data publicly as soon as it is quality-control checked to assure high quality, and data users agree to cite the data producers and to respect the producers’ rights to publish comprehensive analyses of the data in peer-reviewed papers. To assist the community in determining how to cite the data and which analyses the data producers are planning to publish, each grand challenge team will publish a “marker paper” describing the project, its citation policy, and its proposed analyses, at a stage when the initial project planning is mature, but before substantial work has been completed. Following publication of these analyses, the datasets will be made available without publication restrictions under the Creative Commons “by” 3.0 license (http://creativecommons.org/licenses/by/3.0/legalcode). Agreement with these data release terms will be a precondition for community member participation.

Community Software Contributions: All contributions in the form of software code and algorithm implementations that are contributed by the community for iPlant projects shall be accepted utilizing the agreement model and licensing templates adopted from the Apache Software Foundation (ASF). The “Contributor License Agreements” (CLA) (http://www.apache.org/licenses/#clas), defines the terms under which intellectual property has been contributed to the collaborative and would be applicable to individual contributors, academic and industry groups, and their employees. Software applications and documentation that are donated to iPlant projects by individuals or corporations will adhere to ASF “Software Grant Agreement” (SGA) (http://www.apache.org/licenses/#grants).

Appendix 2: Recommendations

Cyberinfrastructure Software and Sustainability Workshop

Top 10 Recommendations

March 26-27, 2009

11. Create a free flow of information between the global open-source community, industry, and academia. Extensible and interoperable CI development should be encouraged. (4.45)

22. The NSF should establish evaluation criteria and funding mechanisms that support software development, release, and life-cycle improvement. This is particularly critical for relatively lower-use software that is essential to the nation's escience objectives but which may not initially have a broad user-base or immediate commercial potential. Funding should be provided to support software development technologies including repositories, user mailing lists, bug-tracking, and testing. (4.35)

44. Funding agencies should award grants to software development after the research phase is done, to sponsor long term sustainable development. (4.33)

1. Support open source community software through investments of time from developers and monies from grants. (4.26)

21. The NSF should permit funding for software incubation, development, and support to be included in future CI proposals, in particular those proposals that are directed at the development of community-oriented CI products such as, but not limited to, innovative parallel libraries, domain-specific grid"stacks," storage management, collaboration tools, visualization (including remote visualization), and portal components. (4.22)

10. Aggressively encourage advance discussions on software interoperability and dissemination. (4.18)

28. The same level of detailed oversight should be used for software licensing/development awards as is used for hardware procurement and installation. (4.17)

50. NSF should invest heavily in infrastructure that facilitates collaboration for researchers and sharing of data tools and results. (4.17)

16. Agencies should strive to create user and developer communities around software, as they are just as important as the actual software development project. (4.13)

43. Software infrastructure projects in particular should use the opensource model. (4.11)

• Support open source community software through investments of time from developers and monies from grants.

• Have local, regional and national centers pay developers to maintain, port and test open source community software.

• Allow open source software developers access to HPC machines.

• Define and standardize a national CI. Fund a permanent group to oversee this and provide high level training to the advocacy groups in EOT.

• Expose the national CI as a platform.

• Build a self-propagating software feedback loop into future CFP.

• Fund hardening extension for projects showing promise and wide adoption, use the Apache foundations mentorship process to ensure best practices.

• Leverage the local expertise of MSI in stabilizing the national CI and making them more useful to the general public. Incentivize this participation through awards of system time and collaborative extensions.

• Leverage virtualization to ease deployment and maximize ROI.

• Aggressively encourage advance discussions on software interoperability and dissemination.

• Create a free flow of information between the global open-source community, industry, and academia. Extensible and interoperable CI development should be encouraged.

• It is time to plan for multidisciplinary science and engineering cyberinfrastructure now to ensure maximum benefit for the research progress as all useful research is multidisciplinary in nature.

• Components: As much as possible, all software developed should consist of components, which at most levels, will allow the insertion or deletion of systems into the architecture.

• EOT should be a key and integral component of all CI.

• Applications and software providers should form partnerships and make the case to the funding agencies that continued maintenance, support, and development of software is necessary to maintain the pace of advancements across scientific domains.

• Agencies should strive to create user and developer communities around software, as they are just as important as the actual software development project.

• The NSF should foster the creation of sustainable grid infrastructure software by focusing on the following areas (1) grid infrastructure standards, (2) interoperability testing, (3) software incubation and (4) software product development.

• The NSF should provide ongoing funding for a diverse CI standards team focused on developing and promulgating standards for the emerging cyberinfrastructure. These standards should cover national, regional and campus cyberinfrastructure.

• Support for standards being developed in the Open Grid Forum should be a major emphasis for NSF.

• The NSF should provide funding for a CI interoperability team. This interoperability team should be charged with cooperatively generating a suite of tests, acceptable test outcomes, and metrics that can be used for standard certification covering national, regional and campus cyberinfrastructure.

• The NSF should permit funding for software incubation, development, and support to be included in future CI proposals, in particular those proposals that are directed at the development of community-oriented CI products such as, but not limited to, innovative parallel libraries, domain-specific grid "stacks," storage management, collaboration tools, visualization (including remote visualization), and portal components.

• The NSF should establish evaluation criteria and funding mechanisms that support software development, release, and life-cycle improvement. This is particularly critical for relatively lower-use software that is essential to the nation's escience objectives but which may not initially have a broad user-base or immediate commercial potential. Funding should be provided to support software development technologies including repositories, user mailing lists, bug-tracking, and testing.

• The NSF should consider developing mechanisms to allow intellectual property value to accrue to some software development activities outside of the open-source community.

• NSF should adopt a foundation community model to support software projects (as demonstrated by the Apache Foundation).

• NSF should actively seek to create software foundations and encourage NSF funded software projects to become a member of such foundations.

• A new paradigm, called data-aware distributed computing, is needed and NSF should give a high priority to support and maintain software in the areas of reliable and efficient data movement, access, and analysis.

• Establish an independent, unbiased national software oversight committee focused on meeting the requirements of a much broader national science community.

• The same level of detailed oversight should be used for software licensing/development awards as is used for hardware procurement and installation.

• Support and Maintenance-Based Business Models (companies offering support and maintenance for community developed software) should be explored.

• Establish a National Software Oversight Committee that

  • investigates, analyzes, and articulates Licensing Costs, Support & Maintenance Costs, Development Costs, Research Impact, Sustainability.

  • organizes and conducts detailed studies on national research software needs.

  • recommends research software to be procured or developed by stake stakeholders.

• Cyberinfrastructure software is needed that manages shared collections for the research teams, manages data distribution for large experiments, manages real-time sensor data streams for observatories, and builds digital libraries of simulations results.

• Financial support for foundations needs to be built into the long term goals of the system development.

• Agencies should provide funding for core development and for applications of the software across scientific domains. Support is also needed for maintenance, hardening, testing and documentation after a core development period. Further support should be through community buy-in and from usage in large-scale funded and self-funded projects.

• Funding should be diversified to ensure multiple approaches are explored. However, no system should be supported that is not capable of interoperability with other solutions.

• NSF should consider that sustaining capabilities and architecting to enable change are often the better choice than sustaining specific software products.

• Designs to develop sustainable scientific CI should separate scientific concerns and operational concerns from the base capabilities and capability interactions required.

• Increase support of activities to support interoperability, for development of reference implementations rather than one-size-fits-all solutions.

• Setting goals for software sustainability and its priorities should be user community driven.

• Take a Cross-agency and cross-government approach when it comes to defining and agreeing on metrics and processes for validation and auditing of software projects.

• Make significant investment in human training to support scientists when using open source software.

• Sustainable software projects should attract sufficient developers and retain those long enough to maintain continuity.

• Software projects should embrace the open source development model early on in the development cycle.

• Software infrastructure projects in particular should use the open source model.

• Funding agencies should award grants to software development after the research phase is done, to sponsor long term sustainable development.

• Agencies should have clear policies and guidelines for how and when project will have to be maintained independently and without funding.

• Community management and support should be achieved with a centralized model, forming small organizations that coordinate activities throughout the user community.

• NSF funding should focus on new development and innovation, but coordinated with the community management organizations to discourage the phenomenon of reinventing the wheel. Additional, separate funding could support outreach and training to adopting institutions and to end user researchers.

• Such organizations would be able to market the cyberinfrastructure software to sectors outside scientific research with common needs (e.g. financial, energy, transportation) to create an even richer ecosystem to support the software going forward.

• Develop and provide infrastructure that helps in all stages of the open source software development process.

• NSF should invest heavily in infrastructure that facilitates collaboration for researchers and sharing of data tools and results.

• If possible, when funding software development leave room in the shared infrastructure for competing ideas to exist.

• We recommend a 10-year program to develop, deploy and operate persistent science gateways with a review at 5 years. Such a program must also shift focus from research explorations to infrastructure deployment and the incorporation of robust system engineering practices this implies.

• We recommend that a sustainable program be initiated through a detailed cross-directorate and perhaps multi-agency study.

• Use RedHat Enterprise MRG.

Appendix 3: Participants

Deb	Agarwal	Lawrence Berkeley National Laboratory	DAAgarwal@lbl.gov
Stan	Ahalt	Ohio Supercomputing Center	ahalt@osc.edu
Gabrielle	Allen	Louisiana State University	allen@bit.csc.lsu.edu
Amy	Apon	Arkansas High Performance Computing Center	aapon@uark.edu
Bill	Barnett	Indiana University	barnettw@indiana.edu
Terry	Benzel	Information Sciences Institute	tbenzel@isi.edu
Martin	Berzins	University of Utah	mb@sci.utah.edu
Subhash	Bhatia	Morehouse College	sbhatia@morehouse.edu
Richard	Blevins	University of Arizona	rblevins@email.arizona.edu
Geoffrey	Brown	Indiana University	geobrown@indiana.edu
John	Campbell	Purdue University	john-campbell@purdue.edu
Ann	Chervenak	Information Sciences Institute	annc@isi.edu
Neil	Chue	Hong Director, OMII-UK N.	ChueHong@omii.ac.uk
Ewa	Deelman	USC/ISI	deelman@isi.edu
Patrick	Dreher	RENCI/North Carolina State University	pdreher@email.unc.edu
Tony	Drummond	Lawrence Berkeley National Laboratory	LADrummond@lbl.gov
Julio	Facelli	University of Utah	julio.facelli@utah.edu
Rhys	Francis	Australian eResearch Infrastructure Council	rhys@pfc.org.au
Dennis	Gannon	Microsoft	gannon@cs.indiana.edu
Dan	Gezelter	Notre Dame	gezelter@nd.edu
Chris	Greer	Networking and Information Technology Research and Development (NITRD)	cgreer@nsf.gov
Robert	Henschel	Indiana University	rhensche@indiana.edu
Joohyun	Kim	Louisiana State University	jhkim@cct.lsu.edu
Gerhard	Klimeck	Purdue University	gekco@purdue.edu
Rich	Knepper	Indiana University	rknepper@indiana.edu
Stacy	Kowalczyk	Indiana University	skowalcz@indiana.edu
Dave	Lambert	Georgetown University	lambertd@georgetown.edu
David	Lifka	Cornell University	lifka@cac.cornell.edu
Peg	Lindenlaub	Indiana University	plinden@indiana.edu
Miron	Livny	University of Wisconsin-Madison	miron@cs.wisc.edu
Pol	Llovet	Montana State University	pol.llovet@gmail.com
Andrew	Lumsdaine	Indiana University	lums@cs.indiana.edu
Clifford	Lynch	Coalition for Networked Information (CNI)	cliff@cni.org
Ruth	Marinshaw	University of North Carolina at Chapel Hill	ruth@email.unc.edu
Suresh	Marru	Indiana University	smarru@indiana.edu
Stuart	Martin	Argonne	smartin@mcs.anl.gov
Scott	McCaulay	Indiana University	smccaula@indiana.edu
Robert	McGrath	NCSA/UIUC	mcgrath@ncsa.uiuc.edu

Kenton	McHenry	NCSA/UIUC	mchenry@uiuc.edu
Elliot	Metsger	Johns Hopkins University	emetsger@jhu.edu
Therese	Miller	Indiana University	millertm@iupui.edu
Makia	Minich	SUN	makia@sun.com
Anastasia	Morrone	Indiana University	amorrone@iupui.edu
Henry	Neeman	University of Oklahoma Information Technology	hneeman@ou.edu
Steven	Newhouse	Enabling Grids for E-sciencE (EGEE)	Steven.Newhouse@cern.ch
Linh	Ngo	University of Arkansas	lngo@uark.edu
Sudhakar	Pamidighantam	NCSA/UIUC	spamidig@ncsa.edu
Marlon	Pierce	Indiana University	mpierce@cs.indiana.edu
Beth	Plale	Indiana University	plale@cs.indiana.edu
Ruth	Pordes	OSG	ruth@fnal.gov
Arcot (Raja)	Rajasekar	University of North Carolina	rajaseka@email.unc.edu
Dick	Repasky	Indiana University	repasky@indiana.edu
Joel	Replogle	Open Grid Forum	replogle@ogf.org
Jennifer	Schopf	NSF	jschopf@nsf.gov
Shava	Smallen	SDSU	ssmallen@sdsc.edu
Brian	Smith	University of New Mexico	carbess@swcp.com
MacKenzie	Smith	MIT Libraries	kenzie@mit.edu
Jef	Spaleta	Fedora Project	jspaleta@gmail.com
Craig	Stewart	Indiana University	stewart@iu.edu
Kevin	Thompson	DHS	kevin.thompson@dhs.gov
John	Towns	NCSA	jtowns@ncsa.uiuc.edu
Greg	Travis	Indiana University	greg@indiana.edu
Susan	Turnbull	DOE	susan.turnbull@ascr.doe.gov
Shel	Waggener	UC Berkeley	shelw@berkeley.edu
Paul	Walk	UKOLN	p.walk@ukoln.ac.uk
Ed	Walker	TACC	ewalker@tacc.utexas.edu
Von	Welch	NCSA	vwelch@ncsa.uiuc.edu
Eric	Wernert	Indiana University	ewernert@indiana.edu
Brad	Wheeler	Indiana University	bwheeler@indiana.edu

Appendix 4: Program
Cyberinfrastructure Software and Sustainability Workshop
Program Agenda, March 26-27, 2009

Wednesday, March 25th		
Check-in and Informal Reception	Bistro Lobby	6-8pm
Thursday, March 26th		
Welcome, goals overview, & brief self introductions by participants – Brad Wheeler	Room 118	8:30am
Software as Cyberinfrastructure: Experience and Perspective from the NSF – Jennifer Schopf	Room 118	9:00am
"An Industry View of the Software Sustainability Challenge" – Brad Wheeler	Room 118	9:45am
Break	10:30am	
Agenda bashing on structure & topics for the breakout sessions – Craig Stewart	Room 118	10:50am
Open Discussion	Room 118	11:00am
Lunch	Bistro Lobby	12:00pm
Breakout sessions & discussion of submitted papers:		1:15pm-3:00pm
How can funding agencies encourage PIs to produce sustainable, reusable software?	Room 118	
How can educators assist in the production of sustainable, reusable software through better education in computer science and software engineering?	Room 226	
How can we develop software so that it is inherently reusable?	Room 232	
What options other than NSF are there for funding software infrastructure?	Room 236	
Break	3:00pm	
Reports from breakout sessions	Room 118	3:15pm
Dennis Gannon "Can we build community application frameworks for science that leverage commercial products?"	Room 118	4:00pm
Break	5:00pm	
Meet & greet, cash bar	Bistro Lobby	5:30pm
Dinner	Scholars Hall	6-8pm
Friday, March 27th		
Neil P. Chue Hong "Cultivating Sustainable Software for Research"	Room 118	8:30am
Clifford Lynch "Software and the Long Haul: Software Design, Adoption, Evolution and Curation in a Cyberinfrastructue Context"	Room 118	9:30am
Break	10:30am	
Breakout sessions & discussion of submitted papers:	11:00am-12:00pm	
Models and recommendations regarding evaluation and adoption of software by VOs	Room 226	
Models and recommendations regarding long term support of software	Room 232	
Discussion of software sustainability, open source software, and value to small schools & MSIs	Room 236	
Box Lunch and discussion	Room 118	12:00pm
Sanjiva Weerawarana – Title TBA	Room 118	1:00pm
Wrap up, identification of areas of consensus, areas of lack thereof, discussion of writing plans	Room 118	1:30pm
Thank yous	Room 118	2:45pm
Adjournment	Room 118	3:00pm

Appendix 5: PowerPoint Slides

An 'Industry' View of the Sustainability Challenge

Dr. Brad Wheeler
Vice President for IT, Dean, & Professor
Indiana University

Professor of Information Systems
IU Kelley School of Business
bwheeler@iu.edu

Sakai

HATHI TRUST
a shared digital repository

I-LIGHT

TeraGrid

REN-ISAC

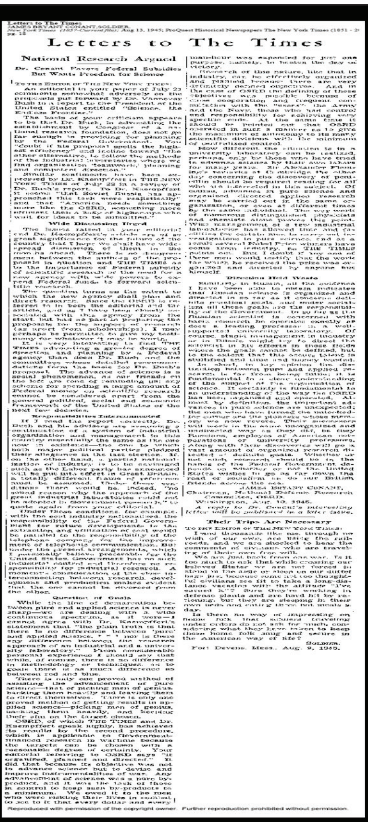
kuali foundation®

GlobalNOC Global Research Network Operations Center

"There is only one proven method of assisting the advancement of pure science – that of picking men of genius, backing them heavily and leaving them to direct themselves."

Letter to the *New York Times,* 13 Aug 1945

James Bryant Conant
President, 1933-1953
Harvard University



Letters to The Times
JAMES BRYANT CONANT,SOLDIER
New York Times (1857-Current file); Aug 13, 1945; ProQuest Historical Newspapers The New York Times (1851 - 2
pg. 18

Letters to The Times

National Research Argued

Dr. Conant Favors Federal Subsidies But Wants Freedom for Science

TO THE EDITOR OF THE NEW YORK TIMES:

An editorial in your paper of July 23 comments somewhat adversely on the proposals put forward by Dr. Vannevar Bush in a report to the President of the United States entitled "Science, the Endless Frontier."

The basis of your criticism appears to be that Dr. Bush, in advocating the establishment by Congress of a national research foundation, does not go far enough in providing for planning by the Federal Government. You "doubt if his proposal spells the highest efficiency" and incline toward "the other alternative, to follow the methods of the industrial laboratories where we find organization 'teamwork,' planning and competent direction."

Similar sentiments have been expressed by Dr. Kaempffert in THE NEW YORK TIMES of July 22 in a review of Dr. Bush's report. To Dr. Kaempffert it seems "that Soviet Russia has approached this task more realistically" and that "America needs something more like the OSRD, something more efficient than a body of highbrows who wait for ideas to be submitted."

Wide Discussion Urged

The issues raised in your editorial and Dr. Kaempffert's article are of so great significance for the future of the country that I hope we shall have widespread discussion of them in the months ahead. There is no disagreement between the authors of the proposals in question and THE TIMES as to the importance of Federal subsidy of scientific research or the need for a new agency with wide powers to expend Federal funds to forward scientific research.

The question turns on the extent to which the new agency shall plan and direct research. Since the OSRD is referred to in both the editorial and the article, and as I have been closely associated with this agency from the start, but had no part in preparing the proposals for the support of research (as apart from scholarships), I may perhaps be permitted to give my testimony for whatever it may be worth.

It is very interesting to find THE TIMES advocating a greater degree of direction and planning by a Federal agency than does Dr. Bush and the committees whose detailed recommendations form the basis for Dr. Bush's proposal. The advance of science is a social phenomenon, as our friends on the left are fond of reminding us; any scheme for spending a large amount of Federal money on scientific research cannot be considered apart from the general political, social and economic framework of the United States of the next few decades.

Responsibilities Interconnected

If I read the report correctly, Dr. Bush and his advisers are assuming a continuation of a system of industrial organization and management in this country essentially the same as the one now in existence—the one to which both major political parties pledged their allegiance in the last election. If, on the other hand, a partial nationalization of industry is to be envisaged such as the Labor party has announced will be put into effect in Great Britain, a totally different frame of reference must be assumed. Under these conditions there would be, indeed, "no sound reason why the approach of the great industrial laboratories could not be adopted in Government research," to quote again from your editorial.

Under these conditions, for example, with the coal industry nationalized, the responsibility of the Federal Government for future developments in the extraction and utilization of coal would be parallel to the responsibility of the telephone company for the improvement of telephonic communication. But under the present arrangements, which I personally believe preferable for the United States, Government has no such industrial control and therefore no responsibility for industrial research. A moment's consideration of the close interconnection between research, development and production makes evident that the one cannot be divorced from the other.

Question of Goals

While the line of demarcation between pure and applied science is never sharp—we are dealing with a wide continuous spectrum, as it were—I cannot agree with Dr. Kaempffert's statement that "the plain truth is that there is no difference between 'pure' and applied science. * * * nor is there any difference between the research approach of an industrial and a university laboratory." From considerable personal experience I should say that while, of course, there is no difference in methodology or techniques, as to goals there is as much difference as between red and blue.

There is only one proved method of assisting the advancement of pure science—that of picking men of genius, backing them heavily and leaving them to direct themselves. There is only one proved method of getting results in applied science—picking men of genius, backing them heavily, and keeping their aim on the target chosen.

OSRD, of which THE TIMES and Dr. Kaempffert speak highly, has achieved its results by the second procedure, which is applicable to Government-financed research in wartime because the targets can be chosen with a reasonable degree of certainty. Your editorial referring to OSRD says "it organized, planned and directed." It did that because its objective was not to advance science but to devise and improve instrumentalities of war. Any advancement of science was a pure by-product, and it was the task of those in control to keep such by-products to a minimum. We owed it to the men who were risking their lives in combat to see to it that every dollar and every man-hour was expended for just one purpose, namely, to hasten the day of victory.

Research of this nature, like that in industry, can be effectively organized and planned because there are very definitely defined objectives. And in the case of OSRD the defining of these objectives was possible because of close cooperation and frequent consultation with the "users," the Army and the Navy, those who had control and responsibility for achieving very specific ends. At the same time it should be pointed out that OSRD operated in such a manner as to give the maximum of autonomy to its many scientific divisions with the minimum of centralized control.

How different the situation is in a university laboratory can be realized, perhaps, only by those who have tried to advance science by their own labors for many years. Sir Alexander Fleming's remarks at Cambridge the other day concerning the discovery of penicillin should be required reading for all who are interested in this subject. Of course, advances in pure science and organized directed applied research may be carried out in the same organization, or even at different times by the same individual. The war work of numerous distinguished physicists and chemists alone proves this point. Wise management of a few industrial laboratories has allowed time and facilities for certain men to carry out investigations in pure science, and as a result several Nobel Prize winners have come from industry, as THE TIMES points out. But I doubt if any one of these men would testify that the work for which he received the prize was organized and directed by anyone but himself.

Direction Held Waste

Similarly in Russia, all the evidence I have been able to obtain indicates that Russian science is organized and directed in so far as it concerns definite practical goals, and under socialism all these goals are the responsibility of the Government. In so far as the Russian scientist is concerned with basic research he operates exactly as does a leading professor in a well-supported university laboratory. Of course, stupid management in industry or in Russia might try to direct the scientist in his efforts in those fields where the goals cannot be defined, but to the extent that this occurs talent is stultified and time and money wasted.

In short, in my opinion the distinction between pure and applied research is far from being futile; it is fundamental to a clear understanding of the subject of the organization of science. It certainly is fundamental to an understanding of the way the OSRD has been organized and operated. Almost by definition, the important advances in pure science are unexpected; the men who have turned the unlooked-for corner are the pioneers whose memory we now revere. Their successors will work in the same unorganized and undirected way as they did, be they Russians, employes of American corporations or university professors. Along with their discoveries will go a vast amount of organized research directed at definite goals. Whether or not such research should be in the hands of the Federal Government depends on whether or not the United States wishes to go as far down the road of socialism as do our British friends across the sea.

JAMES BRYANT CONANT,
Chairman, National Defense Research Committee, OSRD.
Washington, Aug. 10, 1945.

A reply to Dr. Conant's interesting letter will be published in a later issue.

Their Trips Are Necessary

TO THE EDITOR OF THE NEW YORK TIMES:

I and thousands like me, through no wish of our own, are riding the railroads, and we are shocked to hear the comments of civilians who are traveling of their own free will.

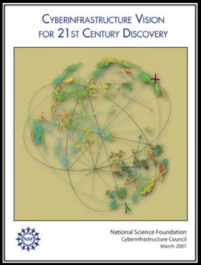
We are just back from one war. Is it too much to ask that while crossing our beloved States we are not forced to stand in the aisle or sleep on our duffel bags just because some not too thoughtful civilians see fit to take a long-distance vacation with the attitude "I've earned it"? Sure they're working in defense plants and are hard hit by rationing, but they are sleeping in their own beds and eating three hot meals a day.

Is there no way of impressing on home folk that soldiers traveling under orders do not ask for much, considering what they have taken to keep these home folk snug and secure in the American way of life?

SOLDIER.
Fort Devens, Mass., Aug. 9, 1945.

"… a new age has dawned in scientific and engineering research, pushed by continuing progress in computing, information, and communication technology, and pulled by the expanding complexity, scope, and scale of today's challenges.

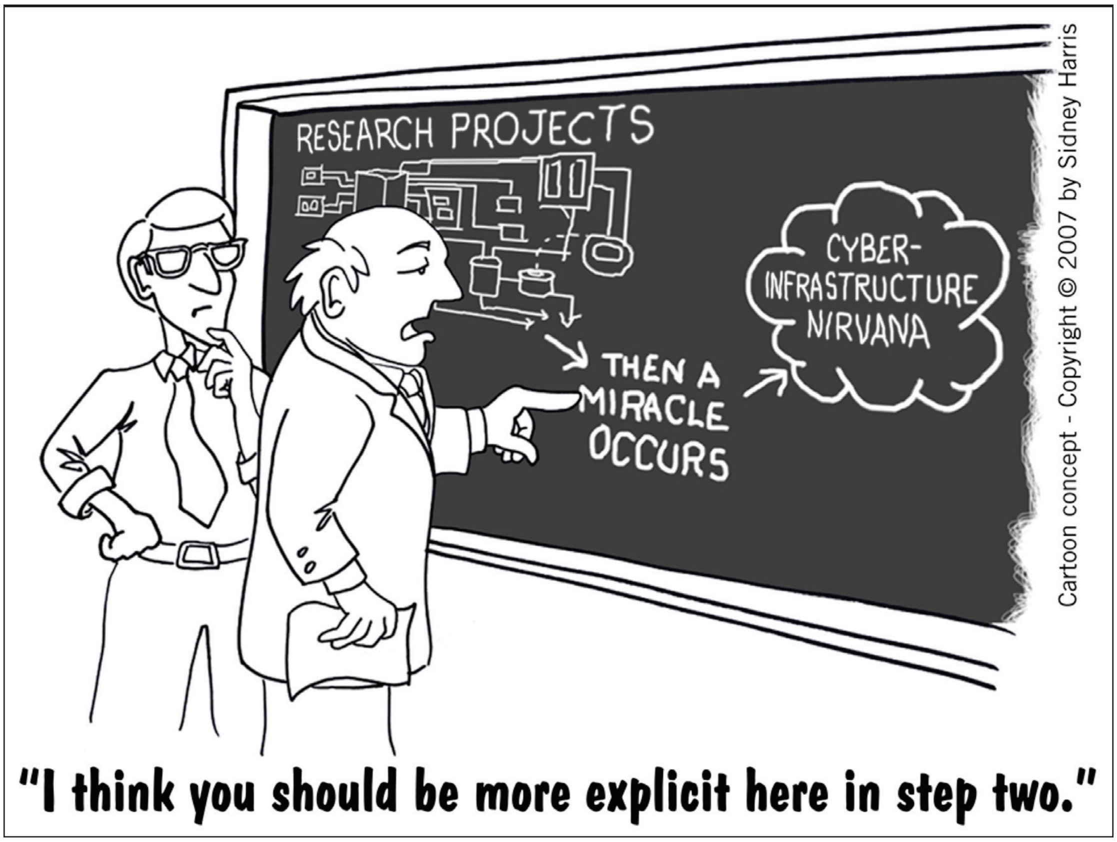
The capacity of this technology has crossed thresholds that now make possible a comprehensive "cyberinfrastructure" on which to build new types of scientific and engineering knowledge environments and organizations…" (p. ES-1)



CYBERINFRASTRUCTURE VISION FOR 21ST CENTURY DISCOVERY

National Science Foundation
Cyberinfrastructure Council
March 2007

Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure, 2003


RESEARCH PROJECTS
THEN A MIRACLE OCCURS
CYBER-INFRASTRUCTURE NIRVANA
Cartoon concept - Copyright © 2007 by Sidney Harris

"I think you should be more explicit here in step two."

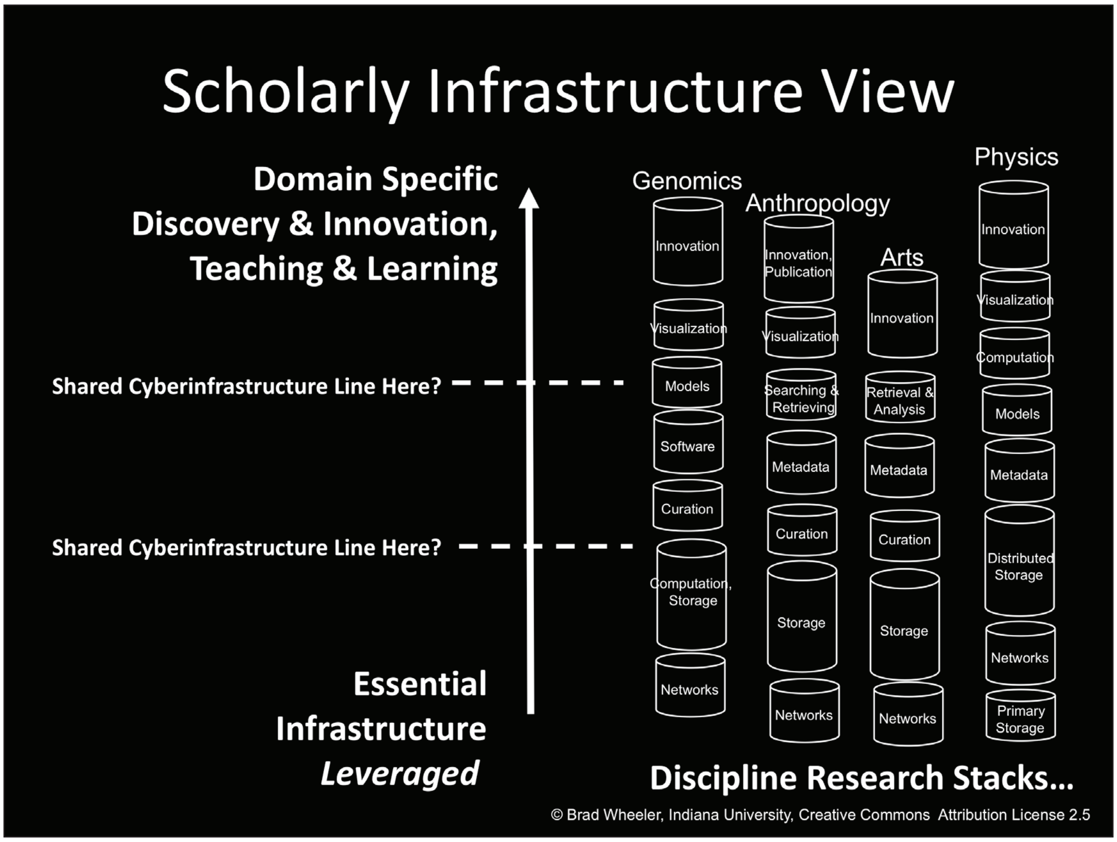
The software artifact is an essential component of

Cyber Infrastructure

Cyber	Infrastructure
of, relating to, or involving computers or computer networks	the system of public works of a country, state, or region ; *also* **:** the resources (as personnel, buildings, or equipment) required for an activity

…and it is a troublesome artifact!

Merriam-Webster Online Dictionary



Scholarly Infrastructure View

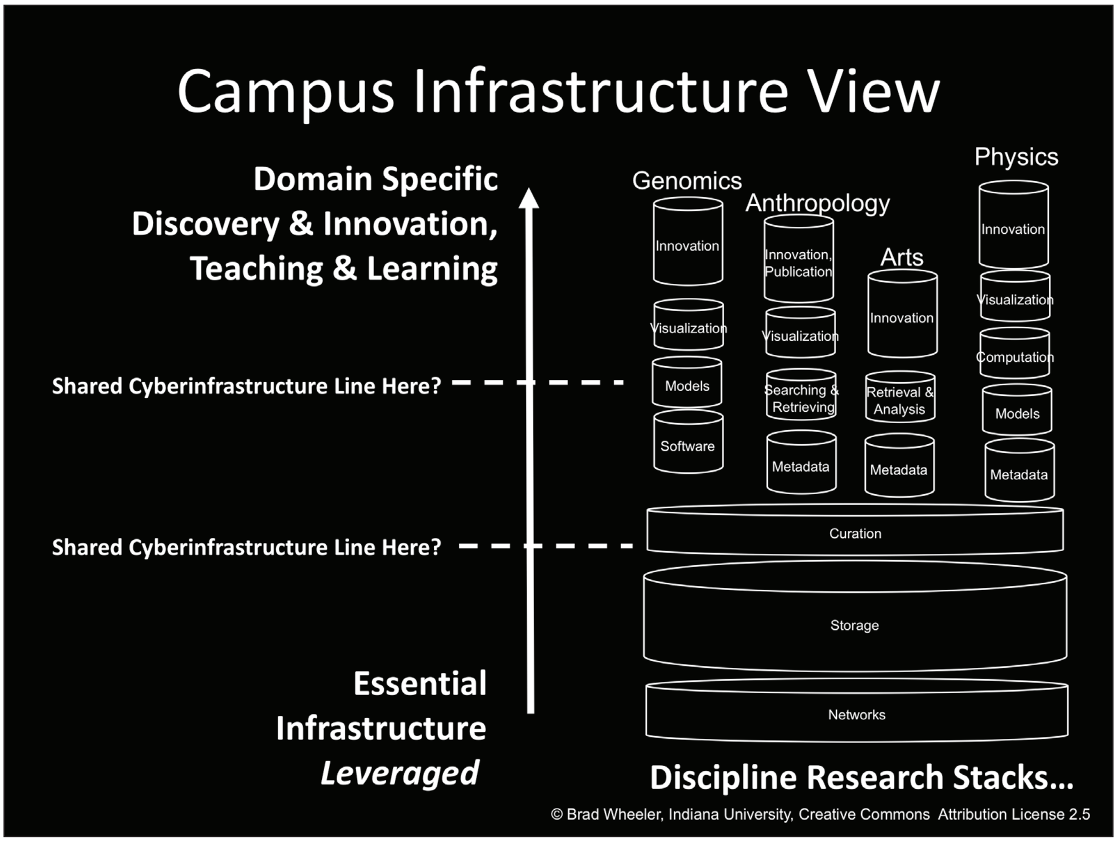
Domain Specific Discovery & Innovation, Teaching & Learning

Shared Cyberinfrastructure Line Here?

Shared Cyberinfrastructure Line Here?

Essential Infrastructure *Leveraged*

Discipline Research Stacks…



Campus Infrastructure View

Domain Specific Discovery & Innovation, Teaching & Learning

Shared Cyberinfrastructure Line Here?

Shared Cyberinfrastructure Line Here?

Essential Infrastructure *Leveraged*

Discipline Research Stacks…

Software Sustainability Chasm

Release .1 – 2.0: Concept & Incubation | Enhancing & Distributing | Documenting QA, Licensing

---------------- Grant Funded Work ----------------

Release 2.1 – 4.0: Concept & Incubation | Enhancing & Distributing | Documenting QA, Licensing

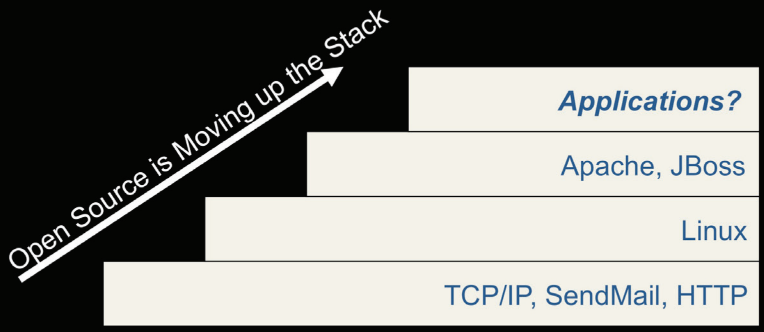
---------------- ???? Funded Work ----------------

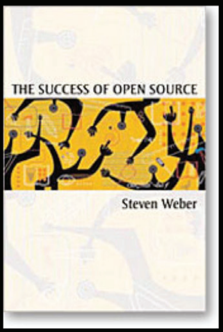
Release 4.1 – R.I.P Concept & Incubation | Enhancing & Distributing | Documenting QA, Licensing

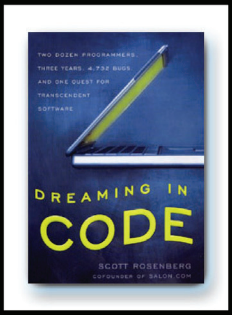
---------------- ???? Funded Work ----------------

Sustaining

Our ‘Derivatives’ Day?







Sustainability Models

Code ←→ Coordination ←→ Community

Code
Coordination
Community

The Code

Linux and Apache have delivered the Code …can CI tools and applications?

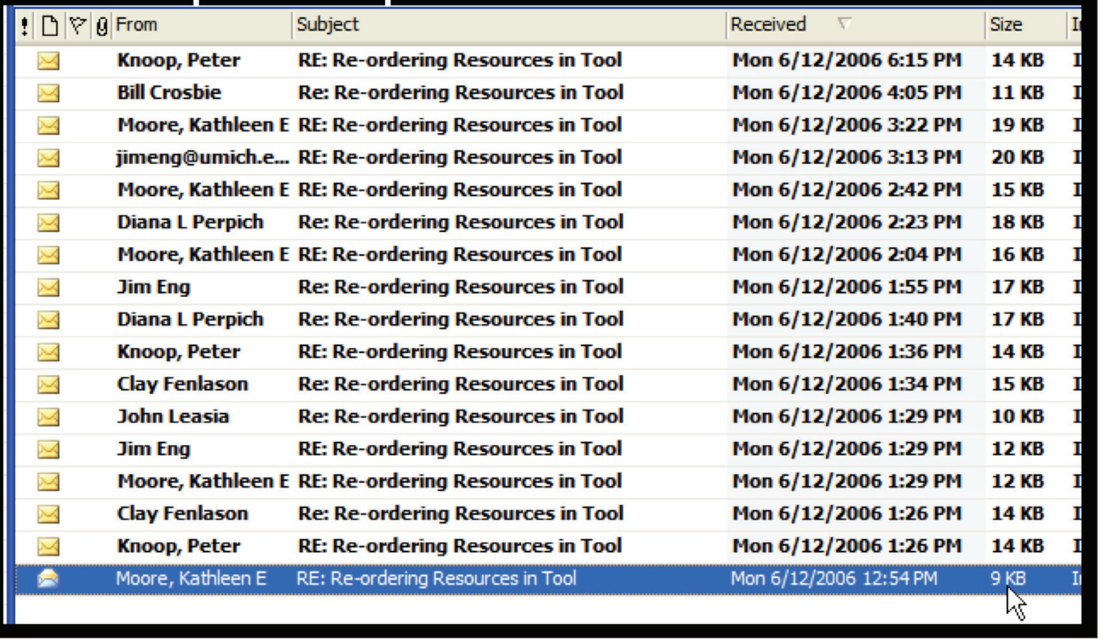
From: The EDUCAUSE CIO Constituent Group Listserv On Behalf of …..
Sent: Thursday, November 18, 2004 4:57 PM

"Dear Mr. Ellison,
…we are not interested in migrating to Collaboration Suite and find ourselves locked out of an upgrade path for our Calendar services. If Oracle is unwilling to sell and support Calendar as a standalone service we will be forced to migrate to one of the competing calendaring systems. This would be unfortunate both because Oracle Calendar is a quality product that has served us well, and because of what it says about Oracle's willingness to listen to and accommodate the needs of its higher education customers.

We are formally requesting that Oracle commit to maintaining the standalone version of Oracle Calendar in rough parity with the collaboration suite calendar component. We would appreciate a response to this request by January 10, 2005, even if it is not technically feasible to deliver the updated stand-alone version in that timeframe.

Sincerely…"

"Shortest Path"
Rapid Requirements Discussions



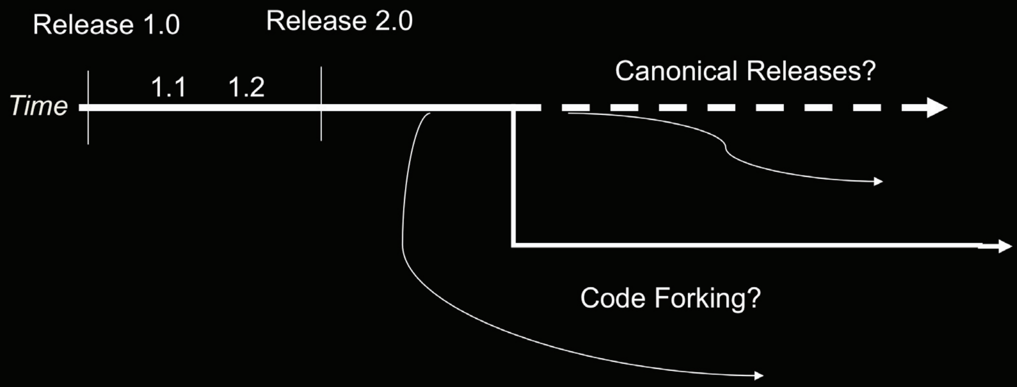
From	Subject	Received	Size	I
Knoop, Peter	RE: Re-ordering Resources in Tool	Mon 6/12/2006 6:15 PM	14 KB	I
Bill Crosbie	Re: Re-ordering Resources in Tool	Mon 6/12/2006 4:05 PM	11 KB	I
Moore, Kathleen E	RE: Re-ordering Resources in Tool	Mon 6/12/2006 3:22 PM	19 KB	I
jimeng@umich.e...	RE: Re-ordering Resources in Tool	Mon 6/12/2006 3:13 PM	20 KB	I
Moore, Kathleen E	RE: Re-ordering Resources in Tool	Mon 6/12/2006 2:42 PM	15 KB	I
Diana L Perpich	Re: Re-ordering Resources in Tool	Mon 6/12/2006 2:23 PM	18 KB	I
Moore, Kathleen E	RE: Re-ordering Resources in Tool	Mon 6/12/2006 2:04 PM	16 KB	I
Jim Eng	Re: Re-ordering Resources in Tool	Mon 6/12/2006 1:55 PM	17 KB	I
Diana L Perpich	Re: Re-ordering Resources in Tool	Mon 6/12/2006 1:40 PM	17 KB	I
Knoop, Peter	RE: Re-ordering Resources in Tool	Mon 6/12/2006 1:36 PM	14 KB	I
Clay Fenlason	Re: Re-ordering Resources in Tool	Mon 6/12/2006 1:34 PM	15 KB	I
John Leasia	Re: Re-ordering Resources in Tool	Mon 6/12/2006 1:29 PM	10 KB	I
Jim Eng	RE: Re-ordering Resources in Tool	Mon 6/12/2006 1:29 PM	12 KB	I
Moore, Kathleen E	RE: Re-ordering Resources in Tool	Mon 6/12/2006 1:29 PM	12 KB	I
Clay Fenlason	Re: Re-ordering Resources in Tool	Mon 6/12/2006 1:26 PM	14 KB	I
Knoop, Peter	RE: Re-ordering Resources in Tool	Mon 6/12/2006 1:26 PM	14 KB	I
Moore, Kathleen E	RE: Re-ordering Resources in Tool	Mon 6/12/2006 12:54 PM	9 KB	I

Code Evolution?

*Questions of **C**oordination*

*Questions of **C**ommunity*

- Quality?
- Feature development?
- Cost (TCO)?



Eben Moglen

- Chairman of the Software Freedom Law Center
- Professor of Law and Legal History at Columbia University Law School
- General Counsel of the Free Software Foundation.

Attack Of The Software Patents!!

Listen to Eben's outstanding keynote speech on systems of production, patents, and the public good:

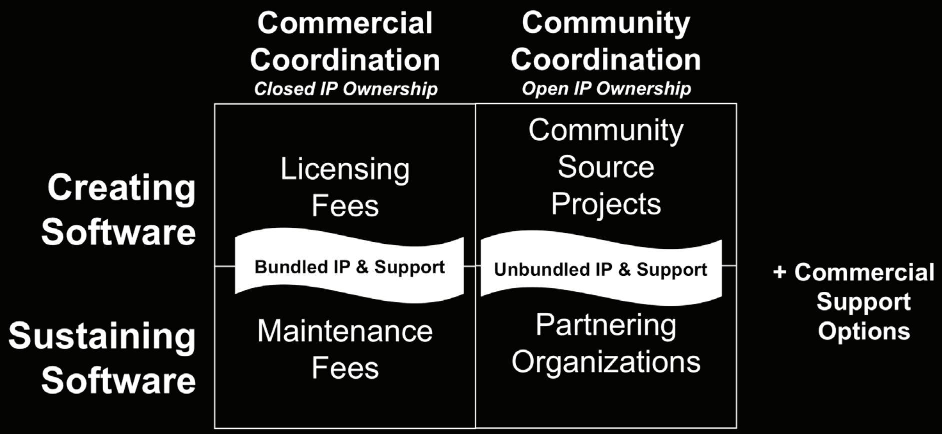
http://issues.sakaiproject.org/confluence/download/attachments/34226/2006_12_06_Keynote_Eben_Moglen.mp3

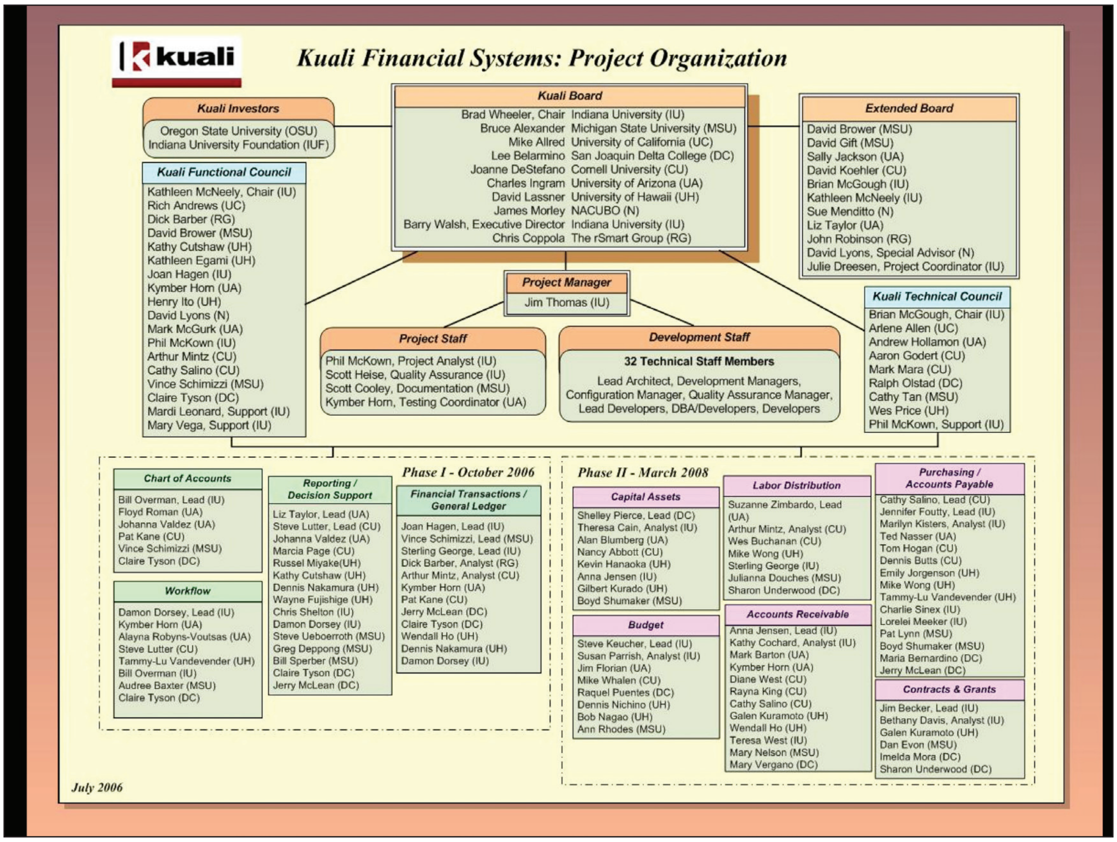
***C**ode*
***C**oordination*
***C**ommunity*

The Coordination

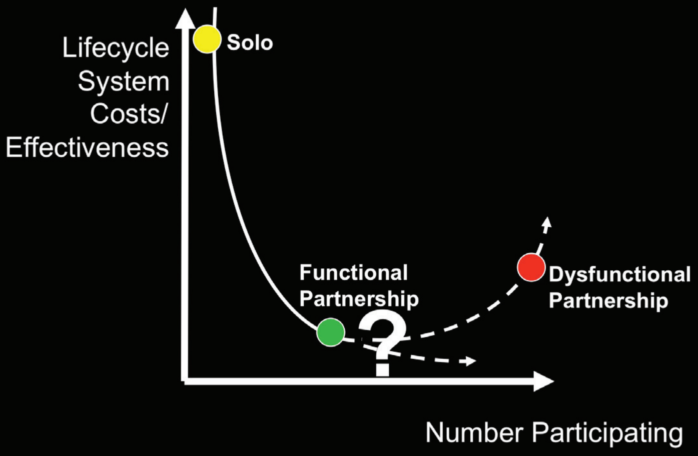
Linux and Apache have demonstrated successful models of coordination… can CI Tools and applications?

Coordination Models





Can we capture economies of scale in software creation and maintenance? *The $Billion question…*



Capturing Industry Leverage:

- •Learning how to partner
- •Synchronizing project/ discipline investments
- •IT architecture discipline
- •Creating effective consortia
- •Common licensing

Code
Coordination
Community

The Community

Linux and Apache have cultivated worldwide, sustaining communities … can applications?

Licensing Shapes a Community

GPL/GNU *"Viral/Copyleft"*

- Code
  - Open source
  - Use without fee
- Derivative works
  - MUST have same license rights
    - Open source
    - Use without fee
    - Allow derivative works

Open-Open *"BSD Style"*

- Code
  - Open source
  - Use without fee
- Derivative Works
  - Any license..

"Birkenstocks and Wingtips: Open Source Licenses," *EDUCAUSE REVIEW,* Jan/Feb 2005.

Licensing

GPL License

"open-open" licensing

Commercial "closed" License

"Birkenstocks"
Free software,
Public good philosophy

"Wingtips"
Capitalist,
Private good philosophy

See http://opensource.org

A hole….Need for a Hybrid Model…
Community Source

"The Pub…the Place Between the Cathedral and the Bazaar"

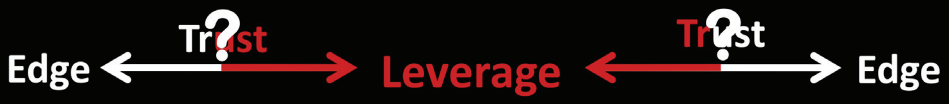
Community Source:

The distinguishing feature of the Community Source Model is that many of the investments of developers' time, design, and project governance come from institutional contributions by colleges, universities, and some commercial firms rather than from individuals. *(wikipedia)*

------------ **Model Archetypes** -----------

	Home Grown	"Techies" Open Source	"The Suits" Open Source	Proprietary Commercial
Code:				
Coordination:				
Community:				

Resource Allocation Problem



Leverage

Discussion Questions

- Is a semi-benevolent dictator essential ? Should there be a basis of power/authority?
- How do distributed communities ensure the "other stuff" gets done…beyond coding?
- Are we in a transition model of software production or moving towards the best path forward?

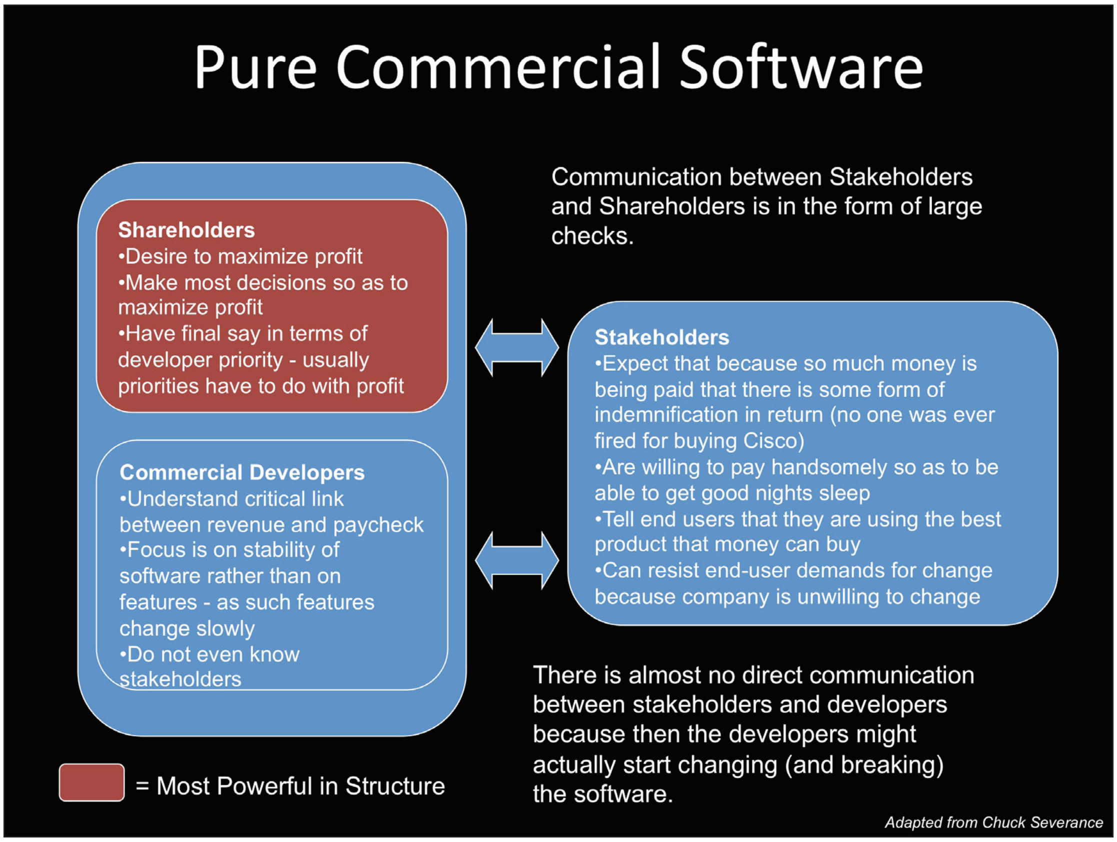
EduCore Concept

An 'Industry' View of the Sustainability Challenge

Dr. Brad Wheeler
Vice President for IT, Dean, & Professor
Indiana University

Professor of Information Systems
IU Kelley School of Business
bwheeler@iu.edu

5 Models



Pure Commercial Software

Shareholders
•Desire to maximize profit
•Make most decisions so as to maximize profit
•Have final say in terms of developer priority - usually priorities have to do with profit

Commercial Developers
•Understand critical link between revenue and paycheck
•Focus is on stability of software rather than on features - as such features change slowly
•Do not even know stakeholders

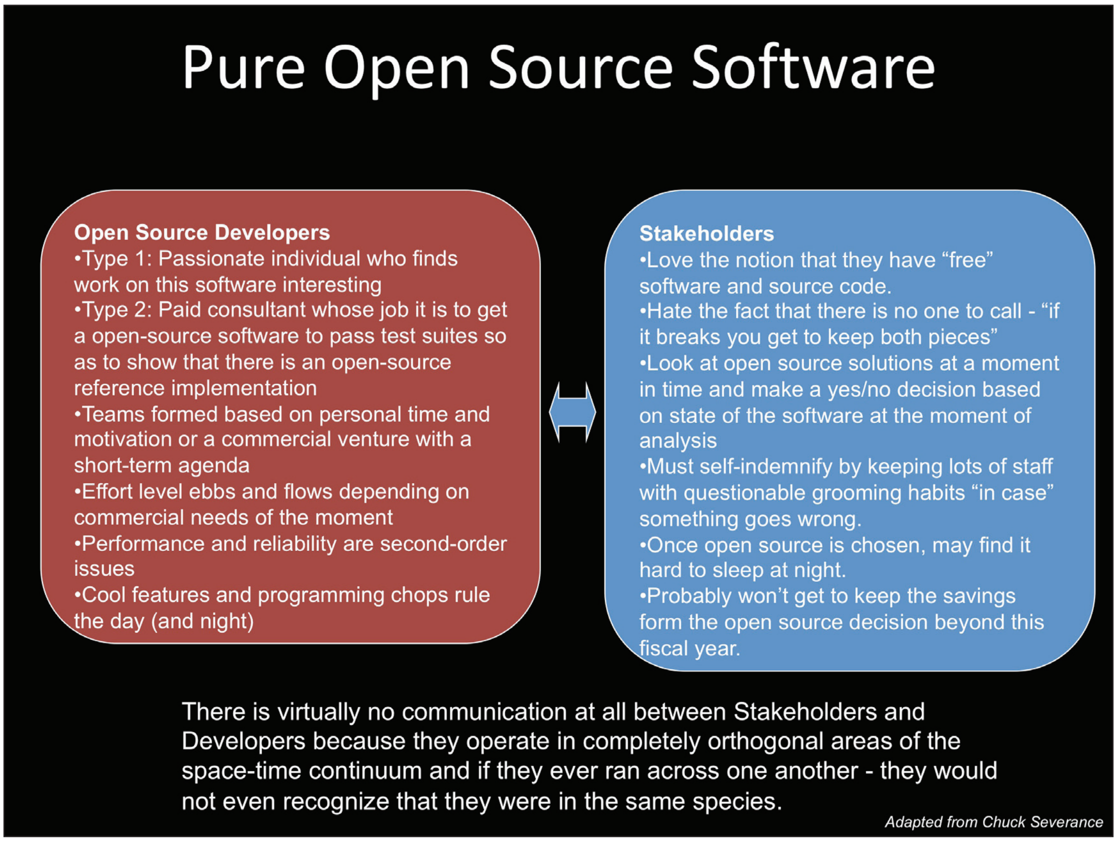
Communication between Stakeholders and Shareholders is in the form of large checks.

Stakeholders
•Expect that because so much money is being paid that there is some form of indemnification in return (no one was ever fired for buying Cisco)
•Are willing to pay handsomely so as to be able to get good nights sleep
•Tell end users that they are using the best product that money can buy
•Can resist end-user demands for change because company is unwilling to change

There is almost no direct communication between stakeholders and developers because then the developers might actually start changing (and breaking) the software.

🟥 = Most Powerful in Structure

Adapted from Chuck Severance



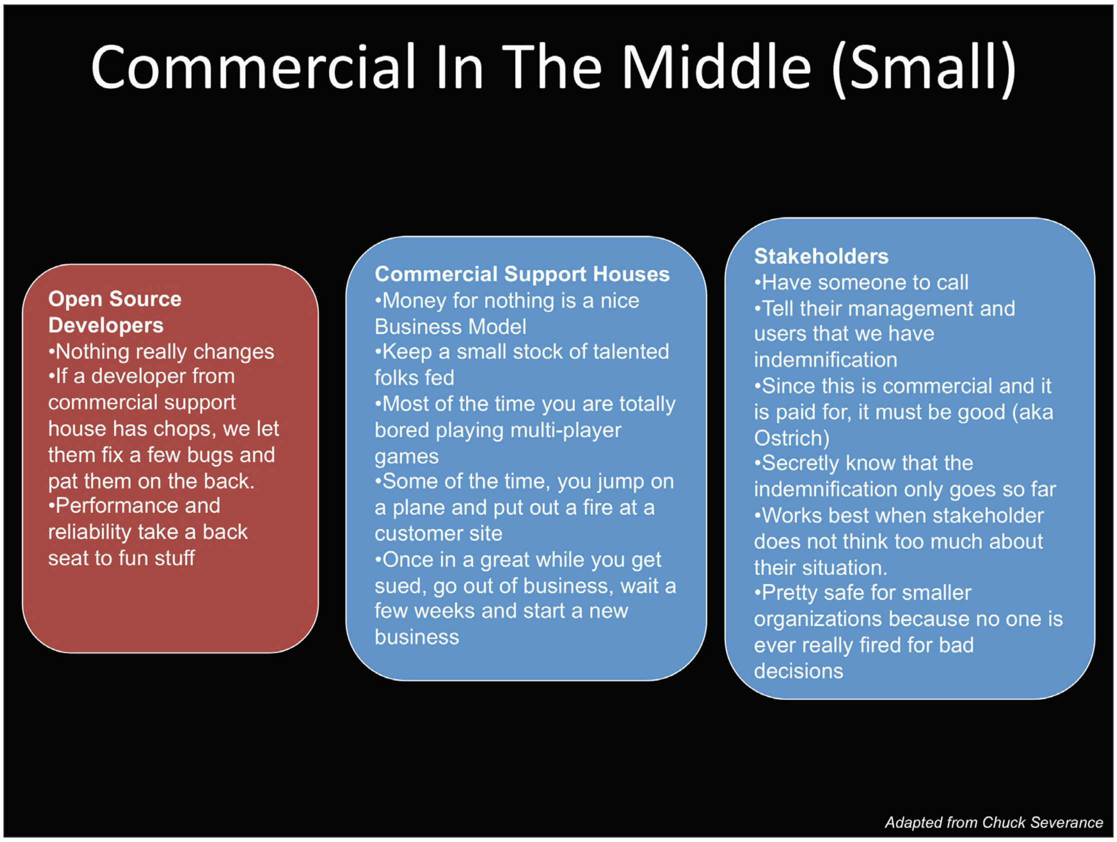
Pure Open Source Software

Open Source Developers
•Type 1: Passionate individual who finds work on this software interesting
•Type 2: Paid consultant whose job it is to get a open-source software to pass test suites so as to show that there is an open-source reference implementation
•Teams formed based on personal time and motivation or a commercial venture with a short-term agenda
•Effort level ebbs and flows depending on commercial needs of the moment
•Performance and reliability are second-order issues
•Cool features and programming chops rule the day (and night)

Stakeholders
•Love the notion that they have “free” software and source code.
•Hate the fact that there is no one to call - “if it breaks you get to keep both pieces”
•Look at open source solutions at a moment in time and make a yes/no decision based on state of the software at the moment of analysis
•Must self-indemnify by keeping lots of staff with questionable grooming habits “in case” something goes wrong.
•Once open source is chosen, may find it hard to sleep at night.
•Probably won’t get to keep the savings form the open source decision beyond this fiscal year.

There is virtually no communication at all between Stakeholders and Developers because they operate in completely orthogonal areas of the space-time continuum and if they ever ran across one another - they would not even recognize that they were in the same species.

Adapted from Chuck Severance



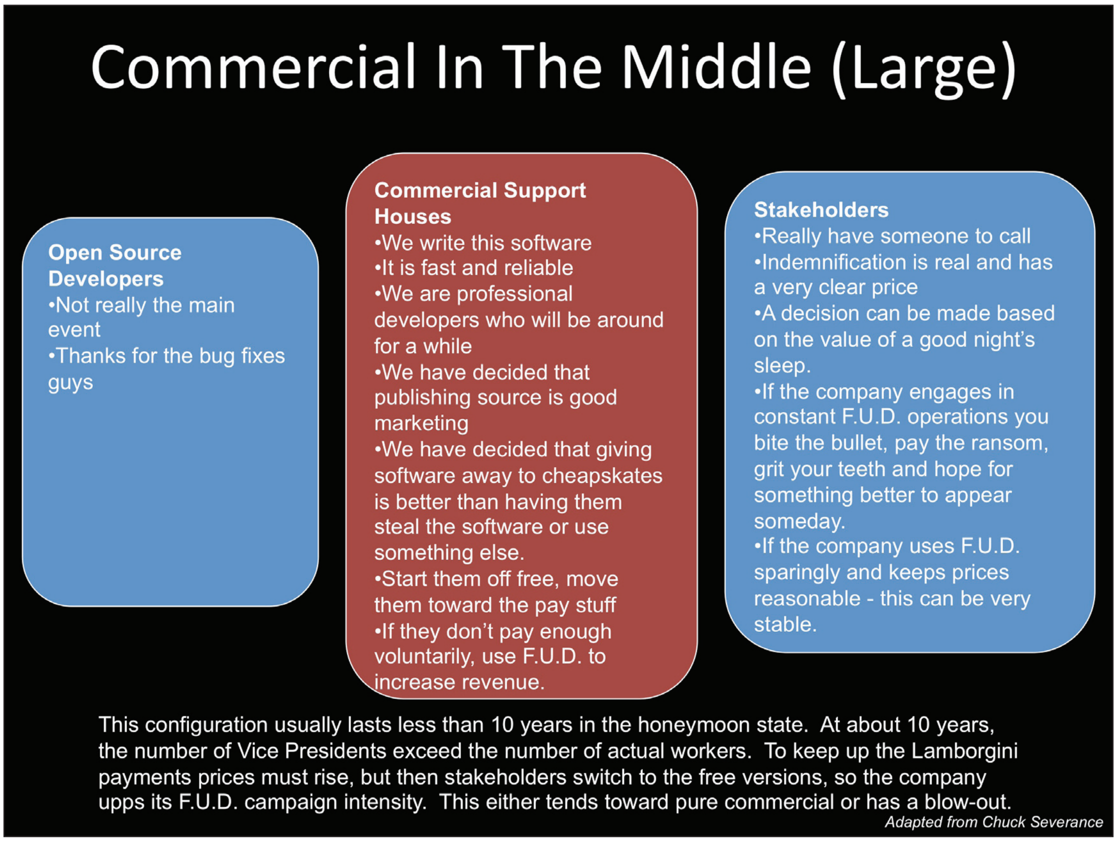
Commercial In The Middle (Small)

Open Source Developers
•Nothing really changes
•If a developer from commercial support house has chops, we let them fix a few bugs and pat them on the back.
•Performance and reliability take a back seat to fun stuff

Commercial Support Houses
•Money for nothing is a nice Business Model
•Keep a small stock of talented folks fed
•Most of the time you are totally bored playing multi-player games
•Some of the time, you jump on a plane and put out a fire at a customer site
•Once in a great while you get sued, go out of business, wait a few weeks and start a new business

Stakeholders
•Have someone to call
•Tell their management and users that we have indemnification
•Since this is commercial and it is paid for, it must be good (aka Ostrich)
•Secretly know that the indemnification only goes so far
•Works best when stakeholder does not think too much about their situation.
•Pretty safe for smaller organizations because no one is ever really fired for bad decisions

Adapted from Chuck Severance



Commercial In The Middle (Large)

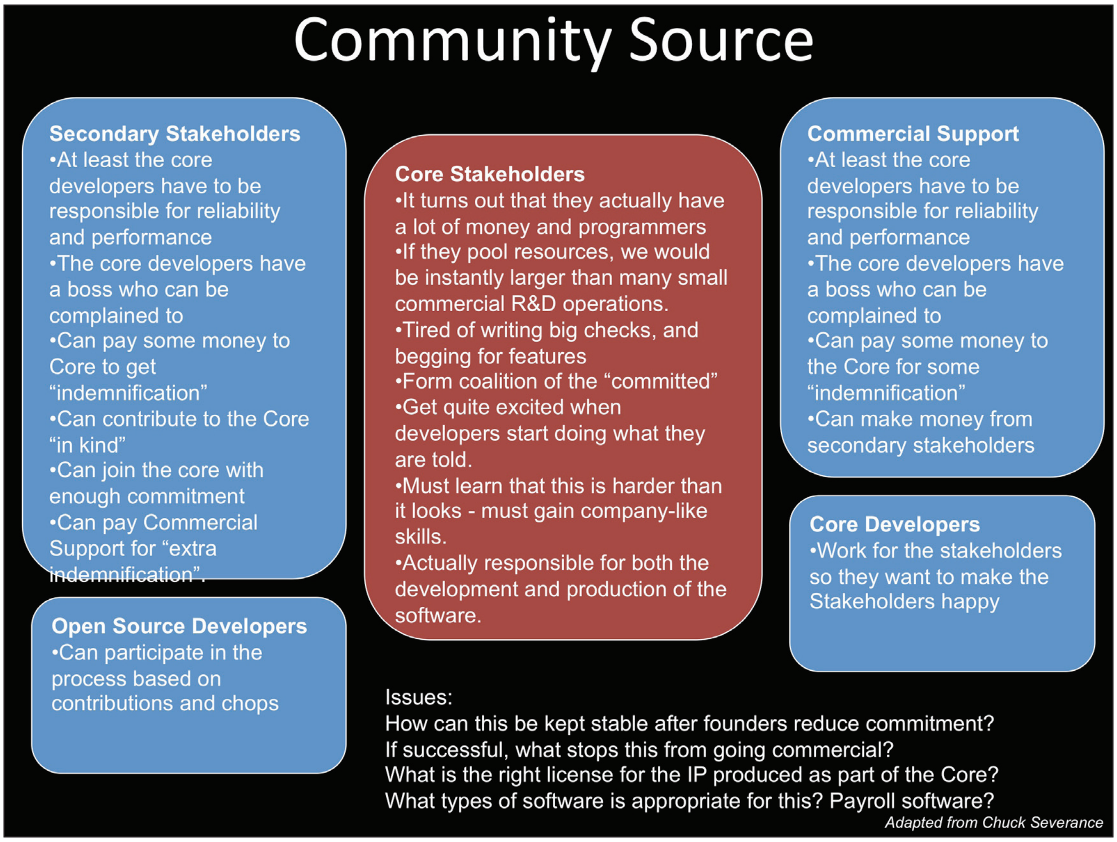
Open Source Developers
•Not really the main event
•Thanks for the bug fixes guys

Commercial Support Houses
•We write this software
•It is fast and reliable
•We are professional developers who will be around for a while
•We have decided that publishing source is good marketing
•We have decided that giving software away to cheapskates is better than having them steal the software or use something else.
•Start them off free, move them toward the pay stuff
•If they don't pay enough voluntarily, use F.U.D. to increase revenue.

Stakeholders
•Really have someone to call
•Indemnification is real and has a very clear price
•A decision can be made based on the value of a good night's sleep.
•If the company engages in constant F.U.D. operations you bite the bullet, pay the ransom, grit your teeth and hope for something better to appear someday.
•If the company uses F.U.D. sparingly and keeps prices reasonable - this can be very stable.

This configuration usually lasts less than 10 years in the honeymoon state. At about 10 years, the number of Vice Presidents exceed the number of actual workers. To keep up the Lamborgini payments prices must rise, but then stakeholders switch to the free versions, so the company upps its F.U.D. campaign intensity. This either tends toward pure commercial or has a blow-out.

Adapted from Chuck Severance



Community Source

Secondary Stakeholders
- At least the core developers have to be responsible for reliability and performance
- The core developers have a boss who can be complained to
- Can pay some money to Core to get "indemnification"
- Can contribute to the Core "in kind"
- Can join the core with enough commitment
- Can pay Commercial Support for "extra indemnification".

Open Source Developers
- Can participate in the process based on contributions and chops

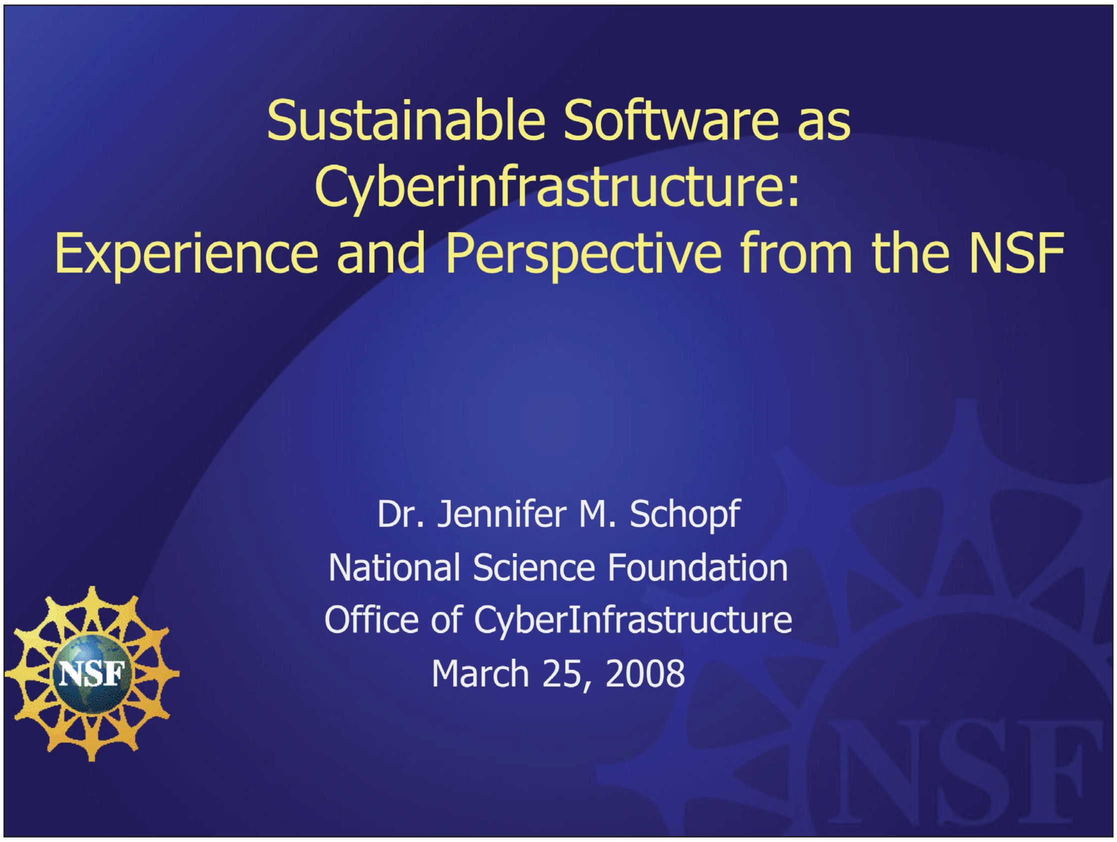
Core Stakeholders
- It turns out that they actually have a lot of money and programmers
- If they pool resources, we would be instantly larger than many small commercial R&D operations.
- Tired of writing big checks, and begging for features
- Form coalition of the "committed"
- Get quite excited when developers start doing what they are told.
- Must learn that this is harder than it looks - must gain company-like skills.
- Actually responsible for both the development and production of the software.

Commercial Support
- At least the core developers have to be responsible for reliability and performance
- The core developers have a boss who can be complained to
- Can pay some money to the Core for some "indemnification"
- Can make money from secondary stakeholders

Core Developers
- Work for the stakeholders so they want to make the Stakeholders happy

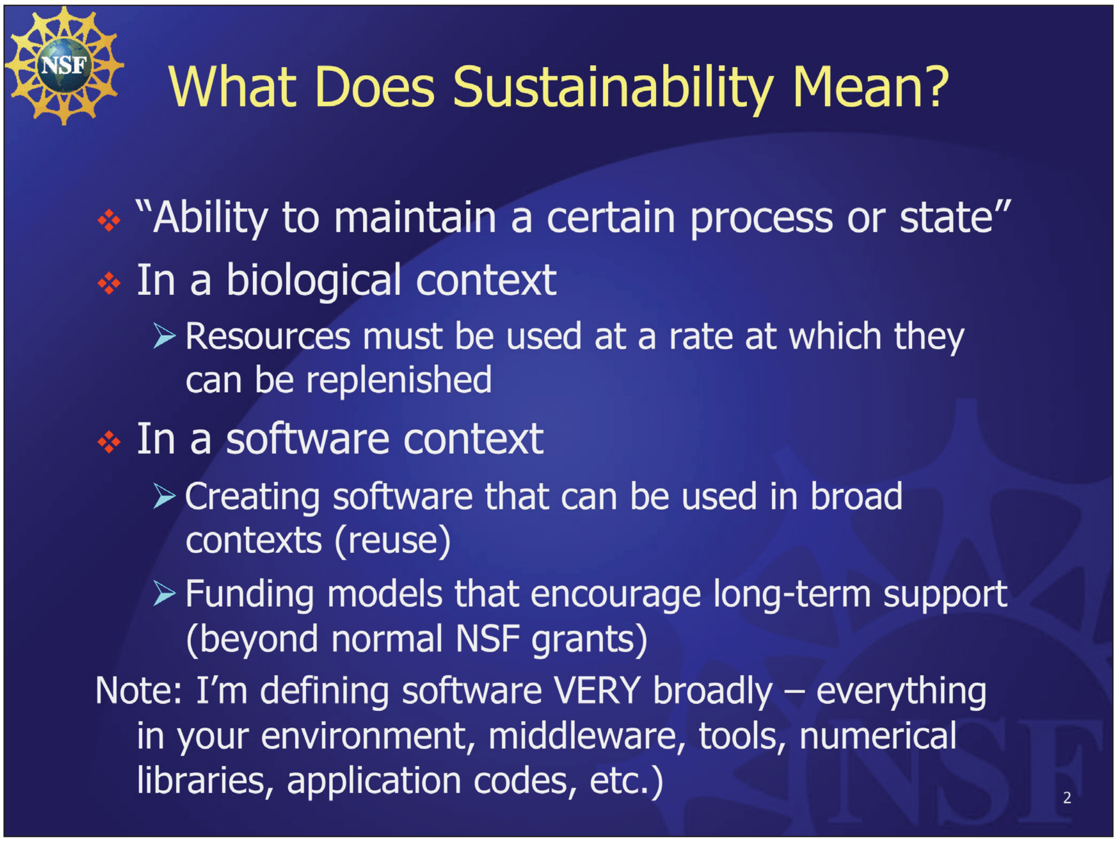
Issues:
How can this be kept stable after founders reduce commitment?
If successful, what stops this from going commercial?
What is the right license for the IP produced as part of the Core?
What types of software is appropriate for this? Payroll software?

Adapted from Chuck Severance



Sustainable Software as Cyberinfrastructure: Experience and Perspective from the NSF

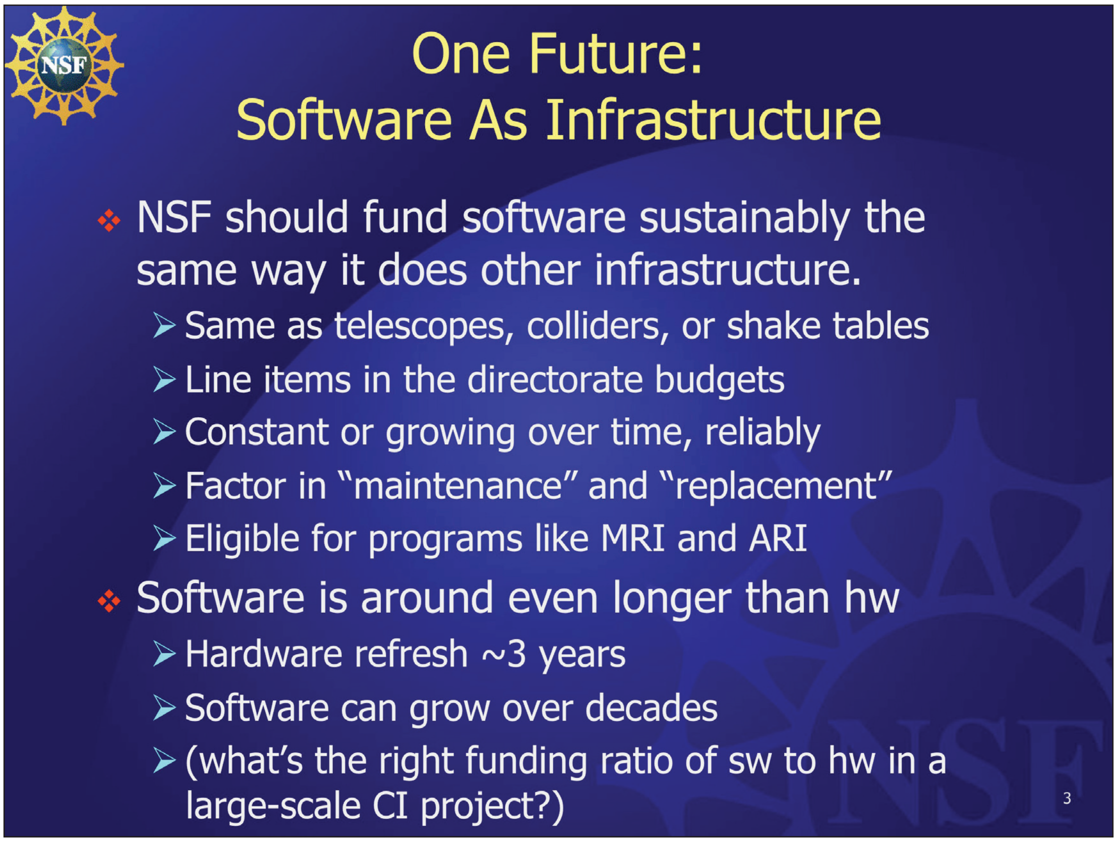
Dr. Jennifer M. Schopf
National Science Foundation
Office of CyberInfrastructure
March 25, 2008



What Does Sustainability Mean?

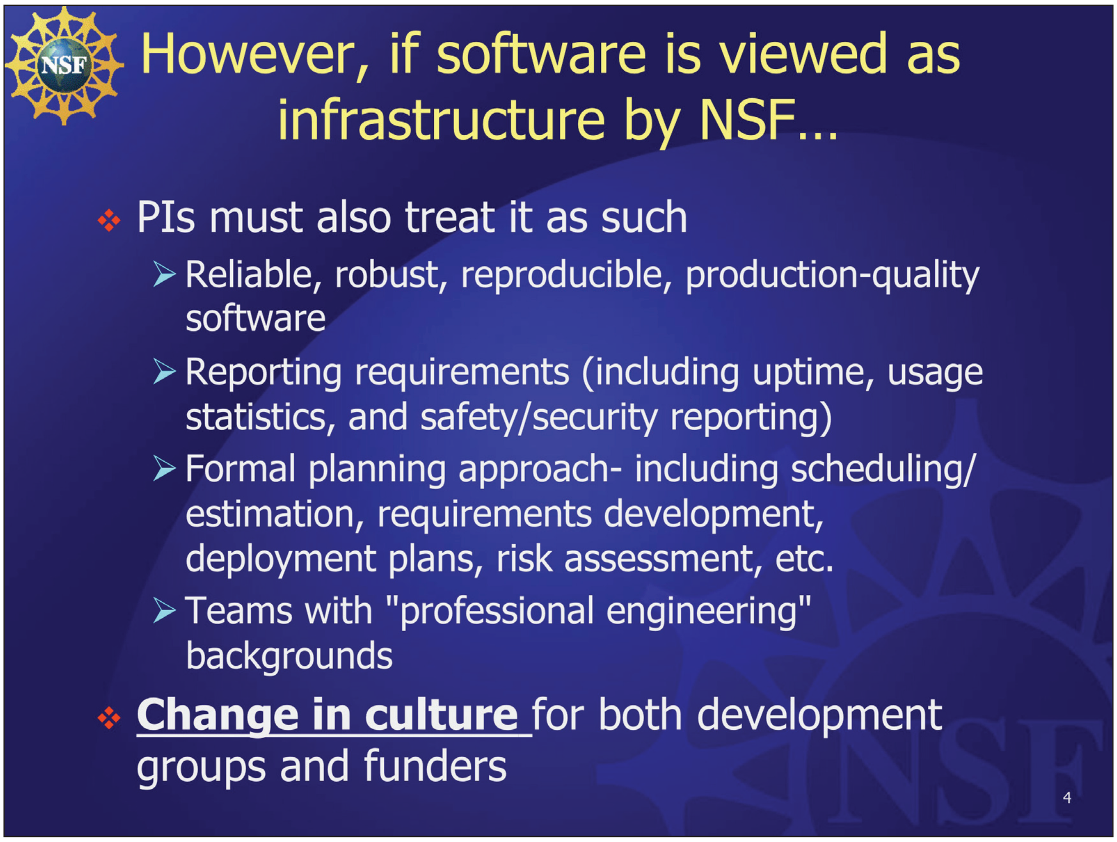
- "Ability to maintain a certain process or state"
- In a biological context
  - Resources must be used at a rate at which they can be replenished
- In a software context
  - Creating software that can be used in broad contexts (reuse)
  - Funding models that encourage long-term support (beyond normal NSF grants)

Note: I'm defining software VERY broadly – everything in your environment, middleware, tools, numerical libraries, application codes, etc.)



One Future: Software As Infrastructure

- NSF should fund software sustainably the same way it does other infrastructure.
  - Same as telescopes, colliders, or shake tables
  - Line items in the directorate budgets
  - Constant or growing over time, reliably
  - Factor in "maintenance" and "replacement"
  - Eligible for programs like MRI and ARI
- Software is around even longer than hw
  - Hardware refresh ~3 years
  - Software can grow over decades
  - (what's the right funding ratio of sw to hw in a large-scale CI project?)



However, if software is viewed as infrastructure by NSF…

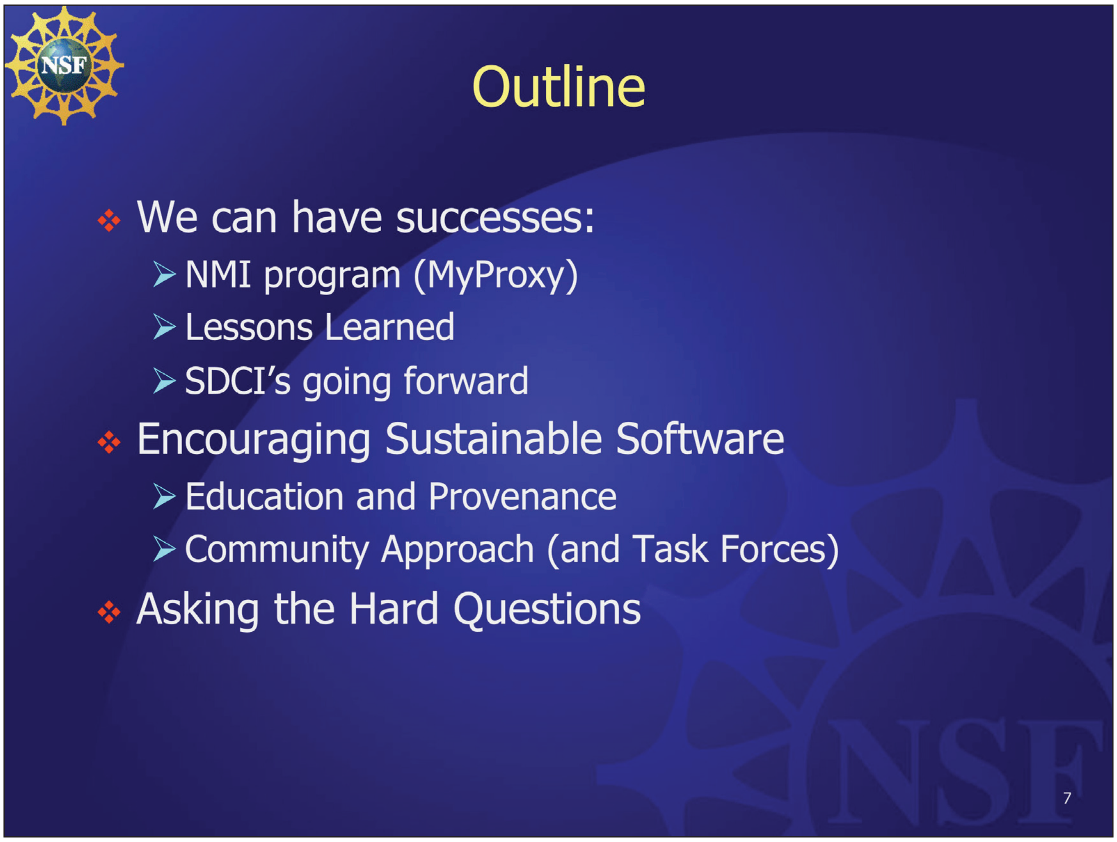
- PIs must also treat it as such
  - Reliable, robust, reproducible, production-quality software
  - Reporting requirements (including uptime, usage statistics, and safety/security reporting)
  - Formal planning approach- including scheduling/ estimation, requirements development, deployment plans, risk assessment, etc.
  - Teams with "professional engineering" backgrounds
- **Change in culture** for both development groups and funders

Note:

- This is not more money
  - More money isn't a solution here
- This is spending the money we have wiser

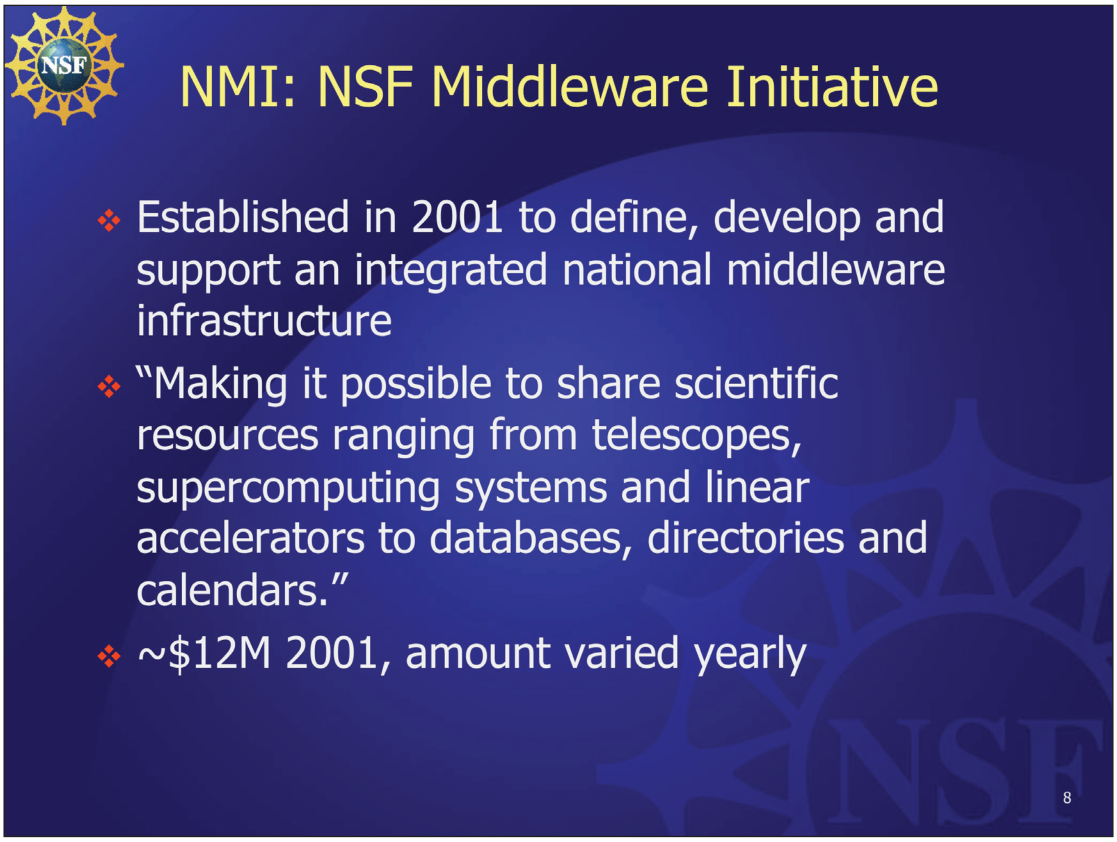
The Question:

- Is it right, is it realizable, and if so, how?
- These are hard questions!
- One way to move forward is for PIs to start down this path
  - Prove to the rest of the community (and funding agencies) this should be true



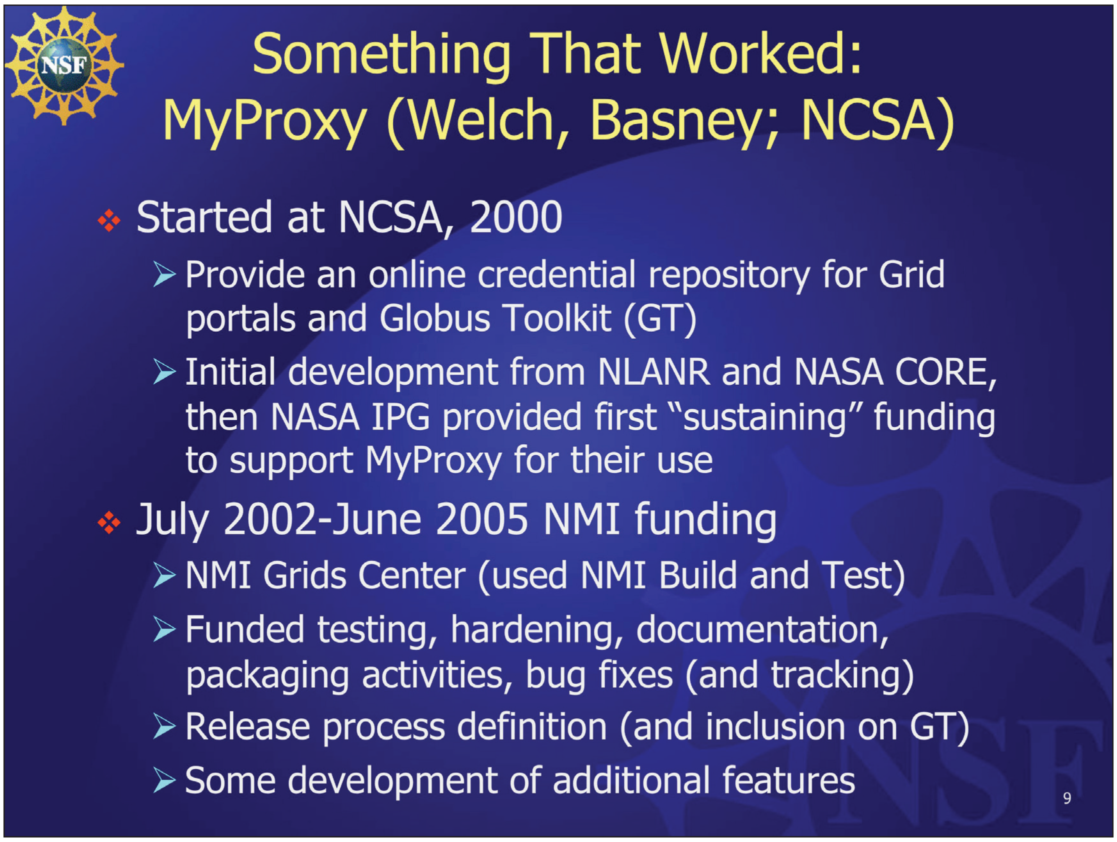
Outline

- We can have successes:
  - NMI program (MyProxy)
  - Lessons Learned
  - SDCI's going forward
- Encouraging Sustainable Software
  - Education and Provenance
  - Community Approach (and Task Forces)
- Asking the Hard Questions



NMI: NSF Middleware Initiative

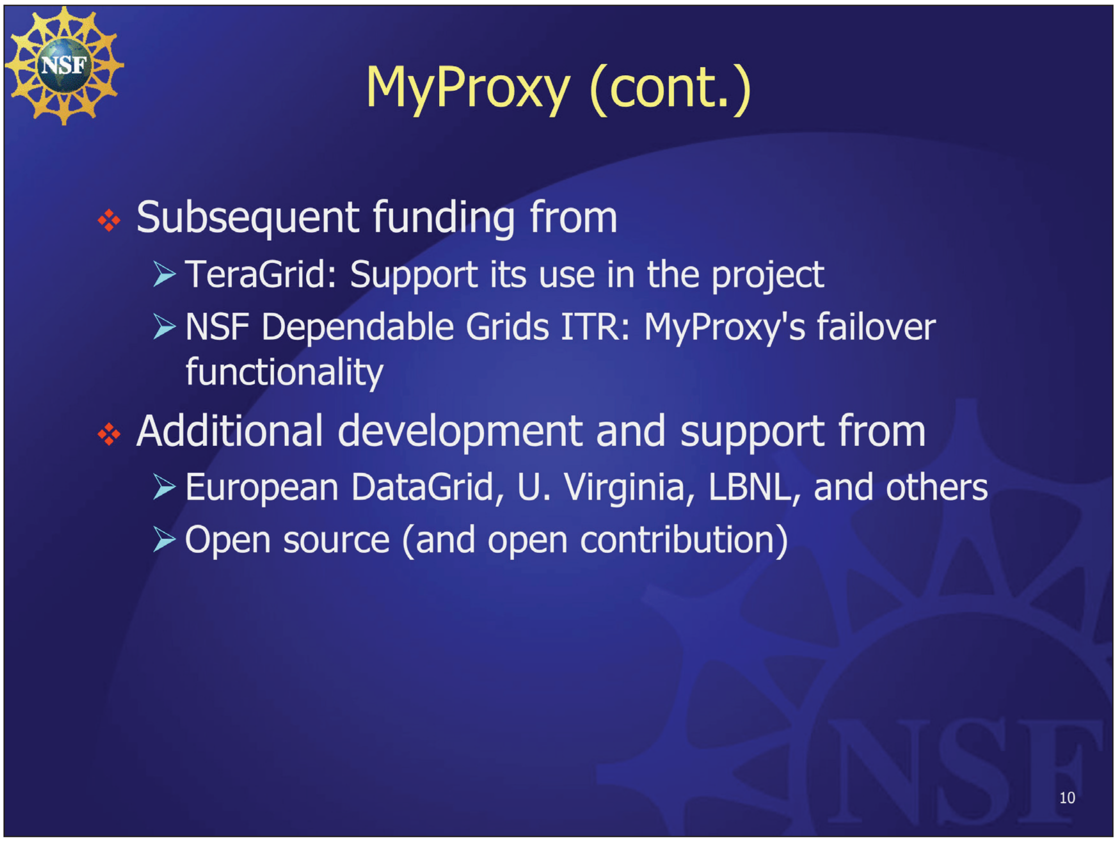
- Established in 2001 to define, develop and support an integrated national middleware infrastructure
- "Making it possible to share scientific resources ranging from telescopes, supercomputing systems and linear accelerators to databases, directories and calendars."
- ~$12M 2001, amount varied yearly



Something That Worked: MyProxy (Welch, Basney; NCSA)

- Started at NCSA, 2000
  - Provide an online credential repository for Grid portals and Globus Toolkit (GT)
  - Initial development from NLANR and NASA CORE, then NASA IPG provided first "sustaining" funding to support MyProxy for their use
- July 2002-June 2005 NMI funding
  - NMI Grids Center (used NMI Build and Test)
  - Funded testing, hardening, documentation, packaging activities, bug fixes (and tracking)
  - Release process definition (and inclusion on GT)
  - Some development of additional features

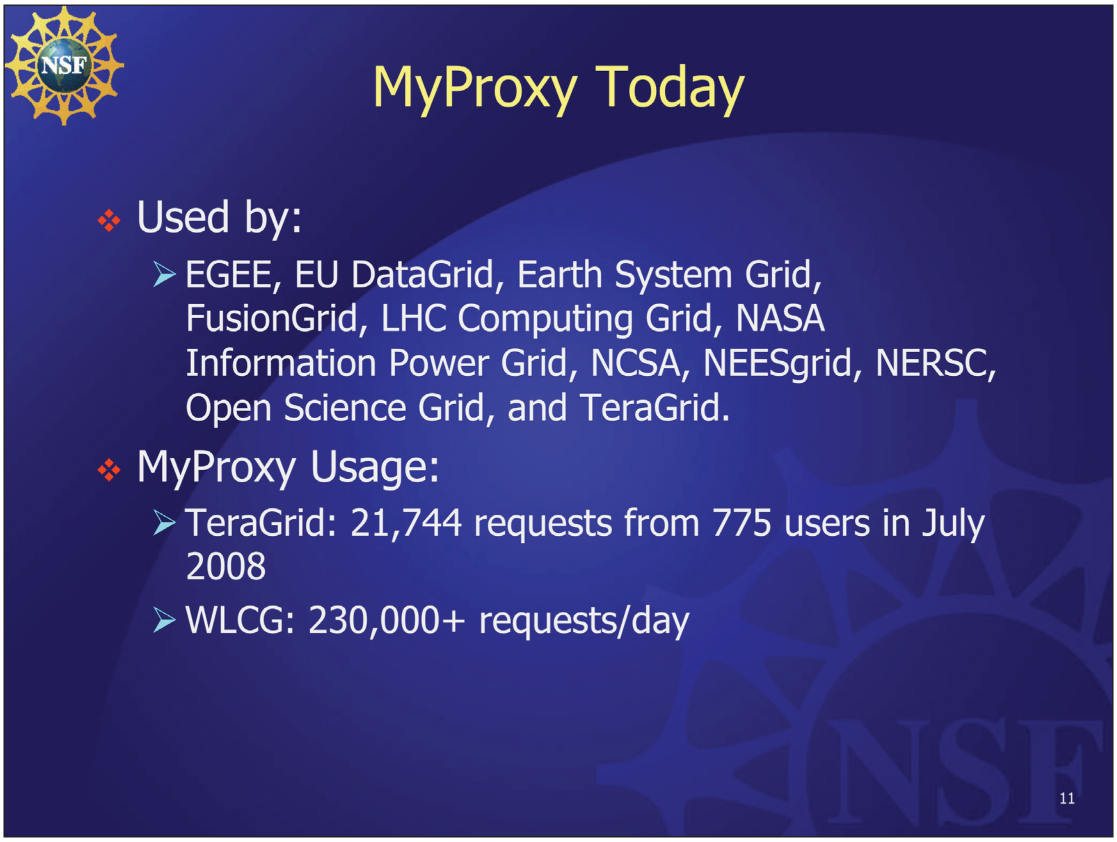
9



MyProxy (cont.)

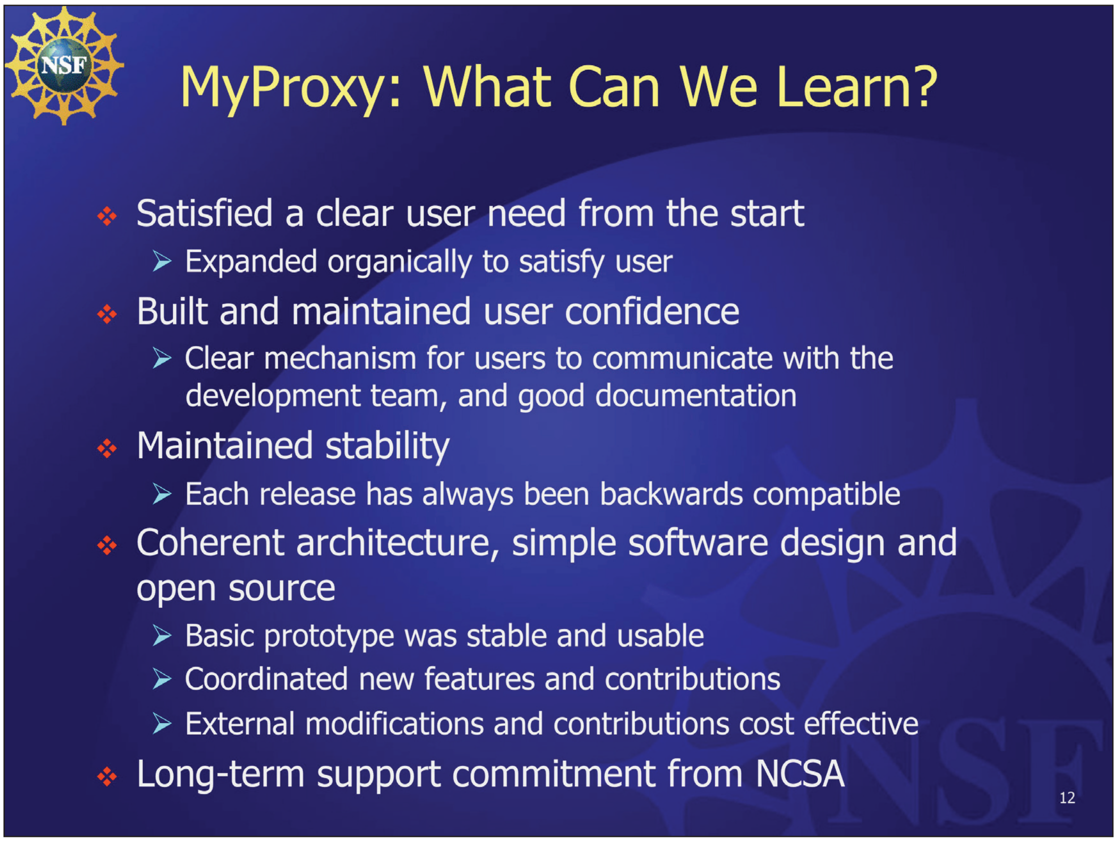
- Subsequent funding from
  - TeraGrid: Support its use in the project
  - NSF Dependable Grids ITR: MyProxy's failover functionality
- Additional development and support from
  - European DataGrid, U. Virginia, LBNL, and others
  - Open source (and open contribution)

10



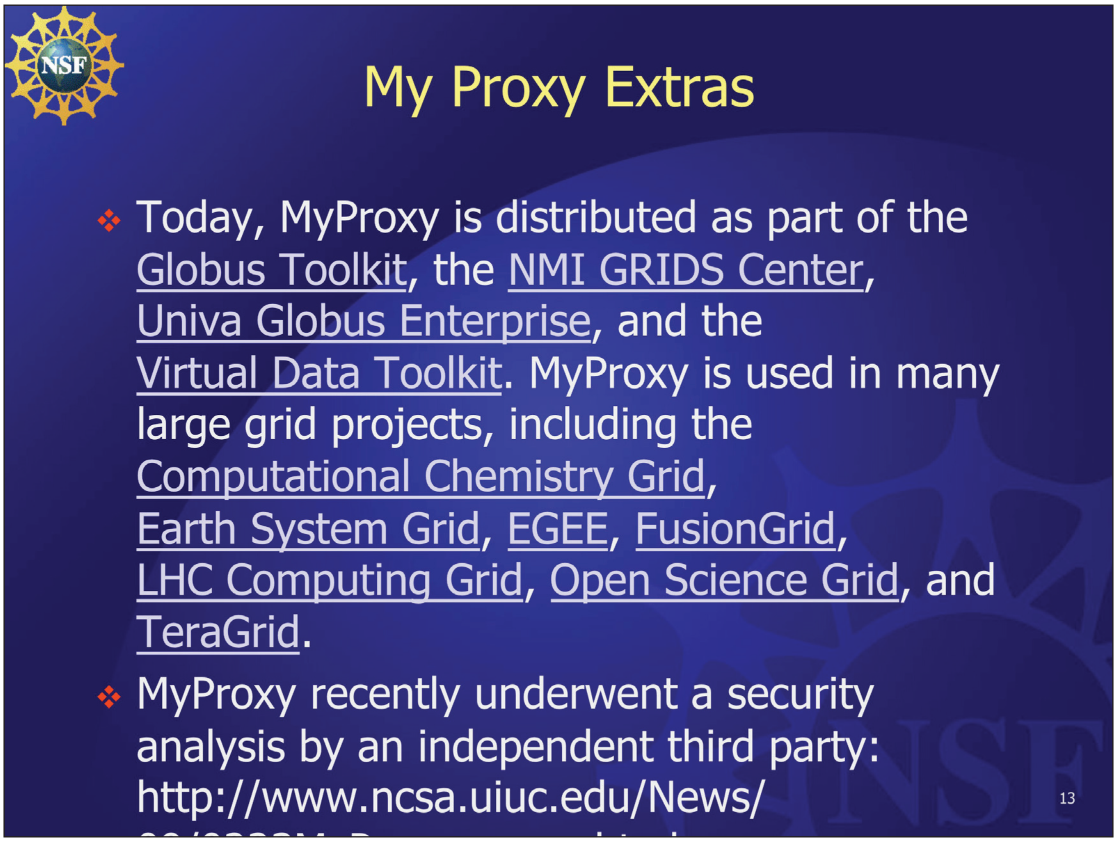
MyProxy Today

- Used by:
  - EGEE, EU DataGrid, Earth System Grid, FusionGrid, LHC Computing Grid, NASA Information Power Grid, NCSA, NEESgrid, NERSC, Open Science Grid, and TeraGrid.
- MyProxy Usage:
  - TeraGrid: 21,744 requests from 775 users in July 2008
  - WLCG: 230,000+ requests/day



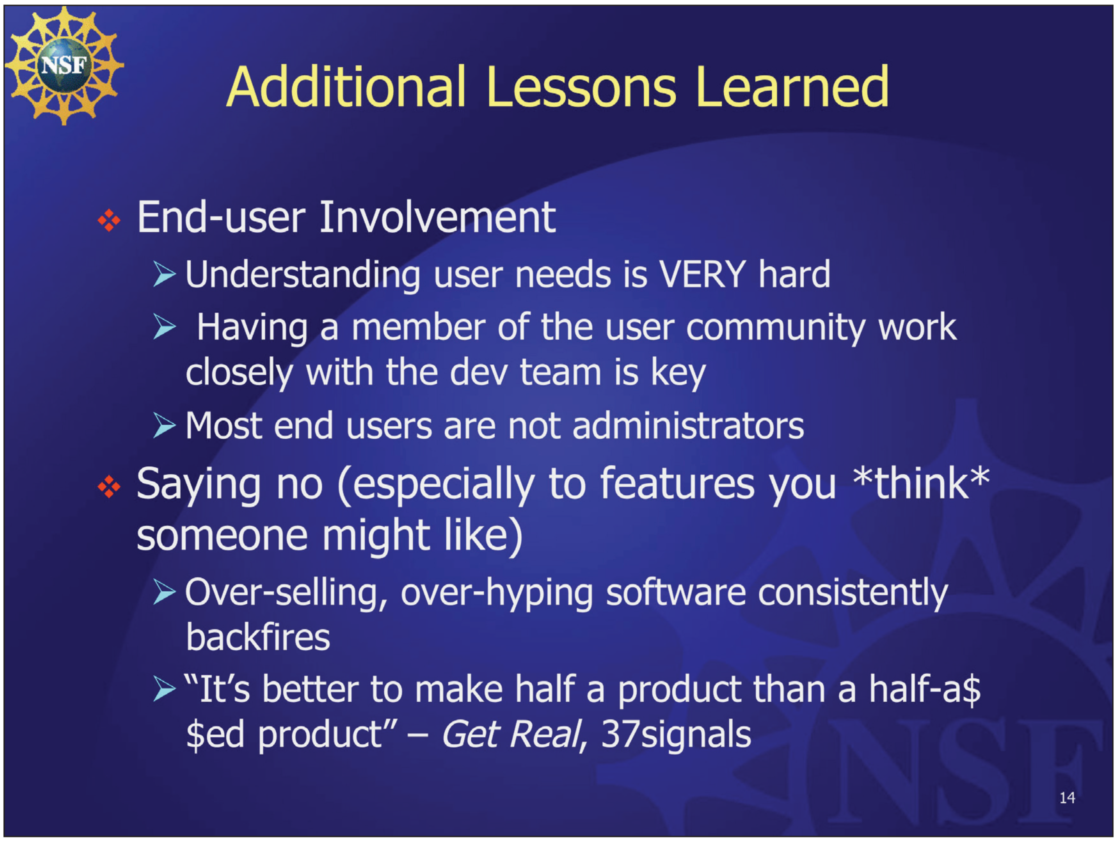
MyProxy: What Can We Learn?

- Satisfied a clear user need from the start
  - Expanded organically to satisfy user
- Built and maintained user confidence
  - Clear mechanism for users to communicate with the development team, and good documentation
- Maintained stability
  - Each release has always been backwards compatible
- Coherent architecture, simple software design and open source
  - Basic prototype was stable and usable
  - Coordinated new features and contributions
  - External modifications and contributions cost effective
- Long-term support commitment from NCSA



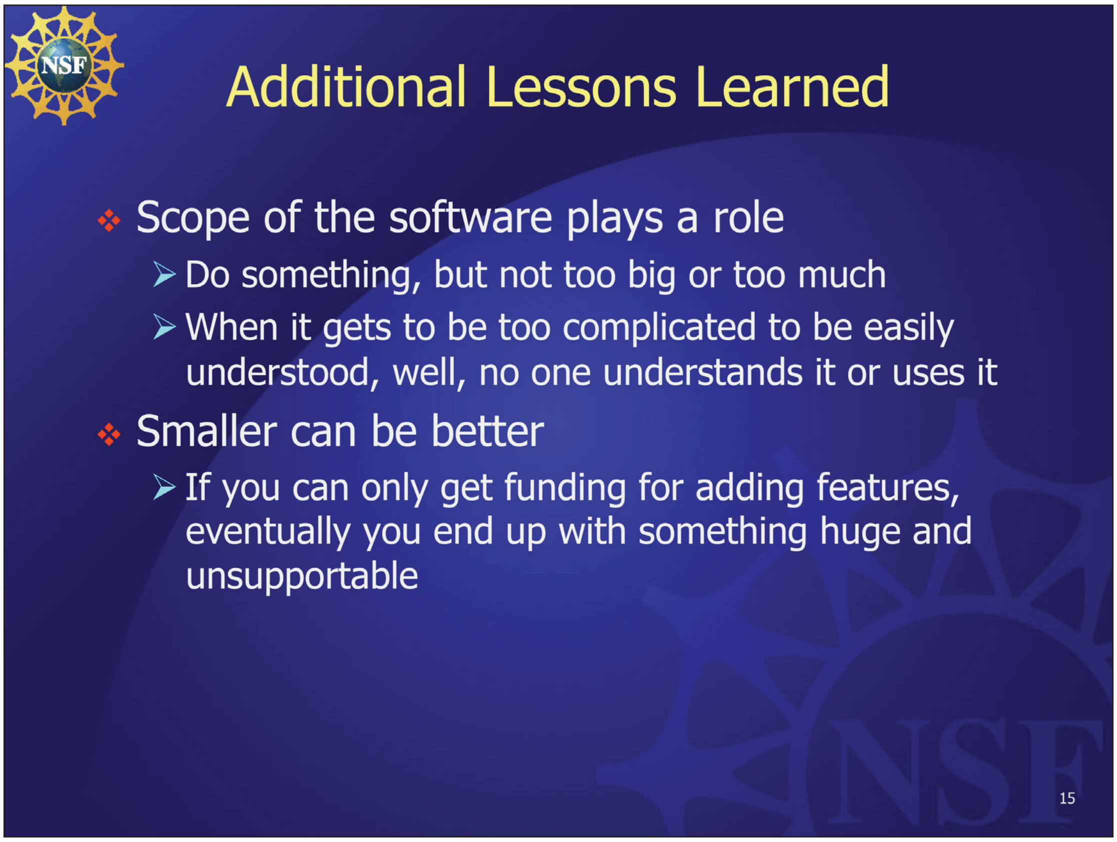
My Proxy Extras

- Today, MyProxy is distributed as part of the Globus Toolkit, the NMI GRIDS Center, Univa Globus Enterprise, and the Virtual Data Toolkit. MyProxy is used in many large grid projects, including the Computational Chemistry Grid, Earth System Grid, EGEE, FusionGrid, LHC Computing Grid, Open Science Grid, and TeraGrid.
- MyProxy recently underwent a security analysis by an independent third party: http://www.ncsa.uiuc.edu/News/



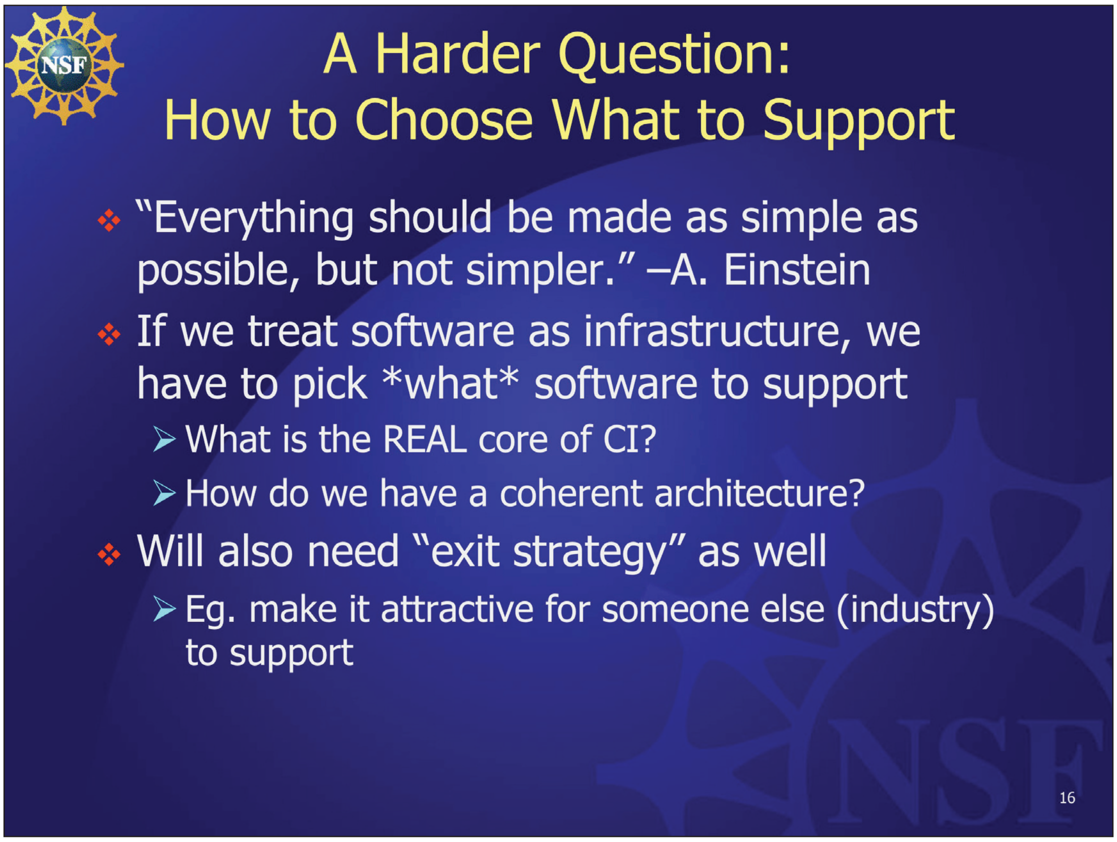
Additional Lessons Learned

- End-user Involvement
  - Understanding user needs is VERY hard
  - Having a member of the user community work closely with the dev team is key
  - Most end users are not administrators
- Saying no (especially to features you *think* someone might like)
  - Over-selling, over-hyping software consistently backfires
  - "It's better to make half a product than a half-a$ $ed product" – *Get Real*, 37signals



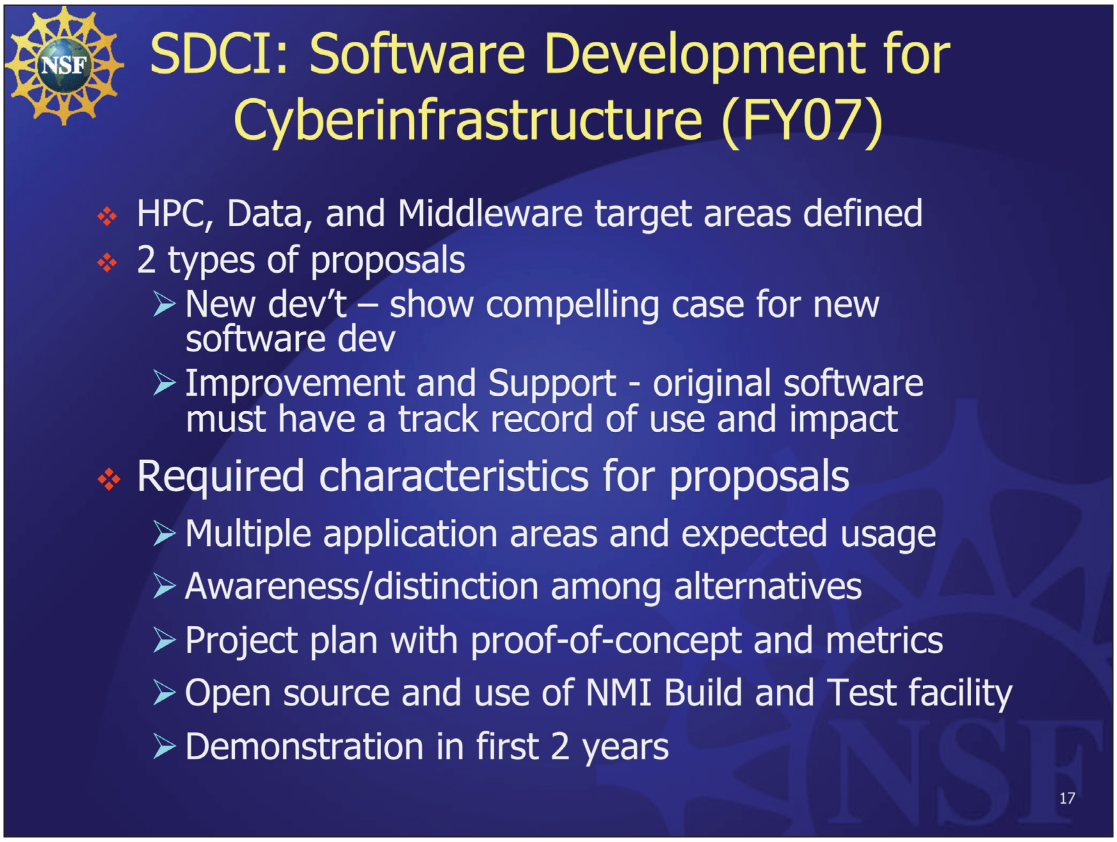
Additional Lessons Learned

- Scope of the software plays a role
  - Do something, but not too big or too much
  - When it gets to be too complicated to be easily understood, well, no one understands it or uses it
- Smaller can be better
  - If you can only get funding for adding features, eventually you end up with something huge and unsupportable



A Harder Question: How to Choose What to Support

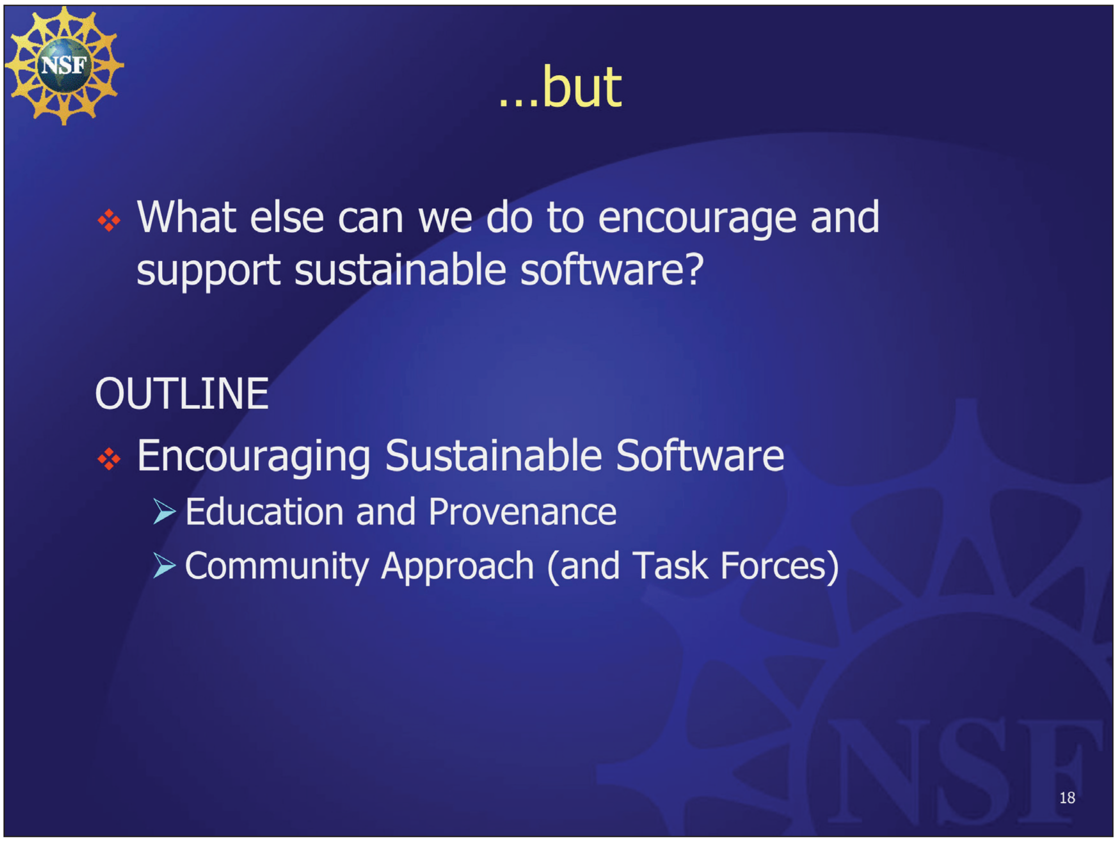
- "Everything should be made as simple as possible, but not simpler." –A. Einstein
- If we treat software as infrastructure, we have to pick *what* software to support
  - What is the REAL core of CI?
  - How do we have a coherent architecture?
- Will also need "exit strategy" as well
  - Eg. make it attractive for someone else (industry) to support



SDCI: Software Development for Cyberinfrastructure (FY07)

- HPC, Data, and Middleware target areas defined
- 2 types of proposals
  - New dev't – show compelling case for new software dev
  - Improvement and Support - original software must have a track record of use and impact
- Required characteristics for proposals
  - Multiple application areas and expected usage
  - Awareness/distinction among alternatives
  - Project plan with proof-of-concept and metrics
  - Open source and use of NMI Build and Test facility
  - Demonstration in first 2 years

17



…but

- What else can we do to encourage and support sustainable software?

OUTLINE

- Encouraging Sustainable Software
  - Education and Provenance
  - Community Approach (and Task Forces)

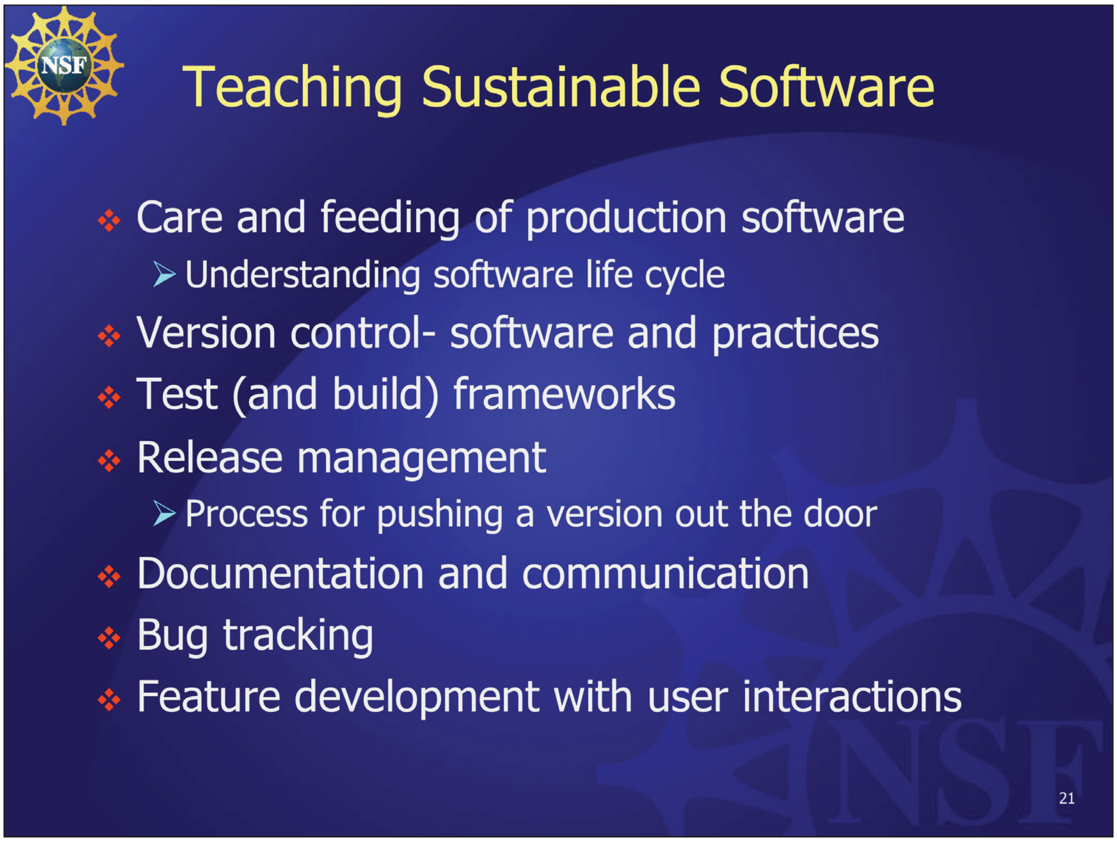
18

Teach Production Software Engineering

- One university's Software Engineering course
  - Systems analysis. Benefit/cost analysis.
  - Project scheduling, management, and control.
  - Requirements Specification document.
  - Development platforms. Prototyping.
  - Human factors. User interface design.
  - Detailed Design document. Configuration management. Program documentation.
  - Documentation. Installation. User training.
  - Software metrics. Cost estimation.
- Individual project developed during course of semester in addition

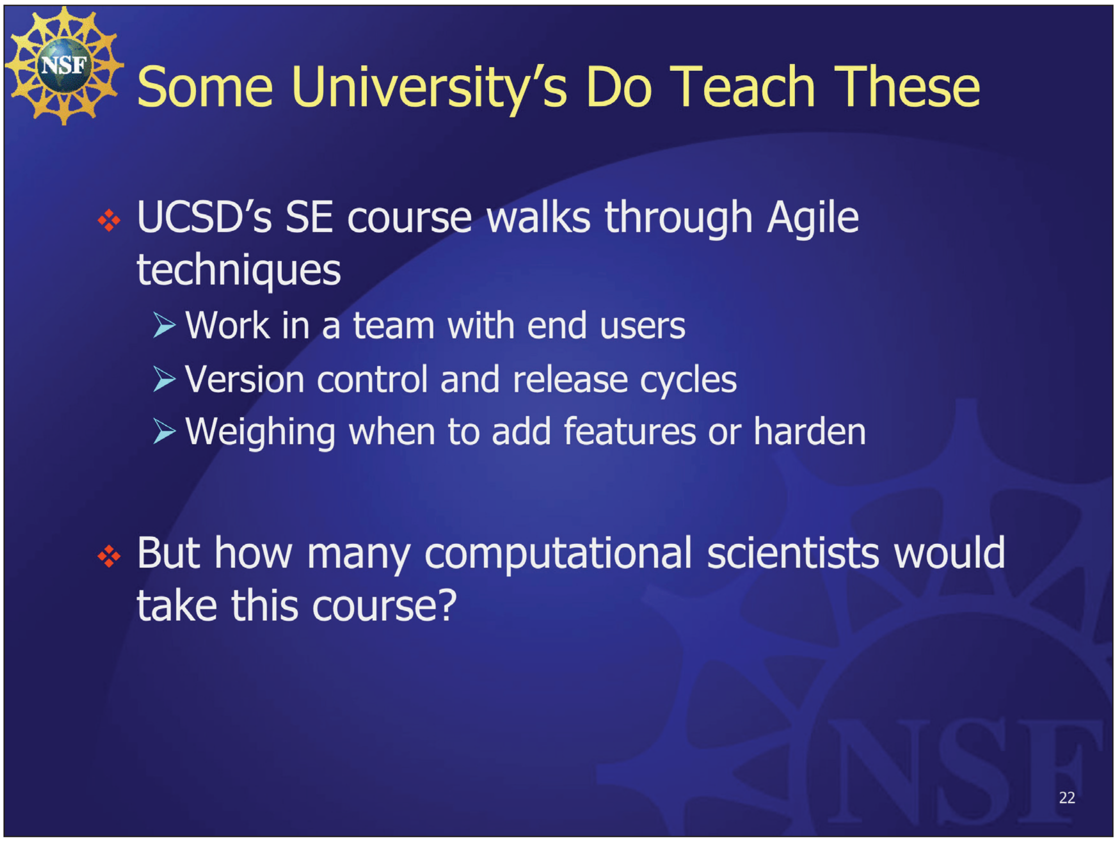
…but

- This doesn't address fundamental issues needed to work in a production environment
  - Working with a team – everything is more complex, from communication to version control
  - Working with end users – changing specifications, documentation, hand holding, negotiation
  - Operations – deployment, performance criteria, interoperabilty
  - Working to deadline – release requirements, tracking bugs, saying No!



Teaching Sustainable Software

- Care and feeding of production software
  - Understanding software life cycle
- Version control- software and practices
- Test (and build) frameworks
- Release management
  - Process for pushing a version out the door
- Documentation and communication
- Bug tracking
- Feature development with user interactions



Some University's Do Teach These

- UCSD's SE course walks through Agile techniques
  - Work in a team with end users
  - Version control and release cycles
  - Weighing when to add features or harden

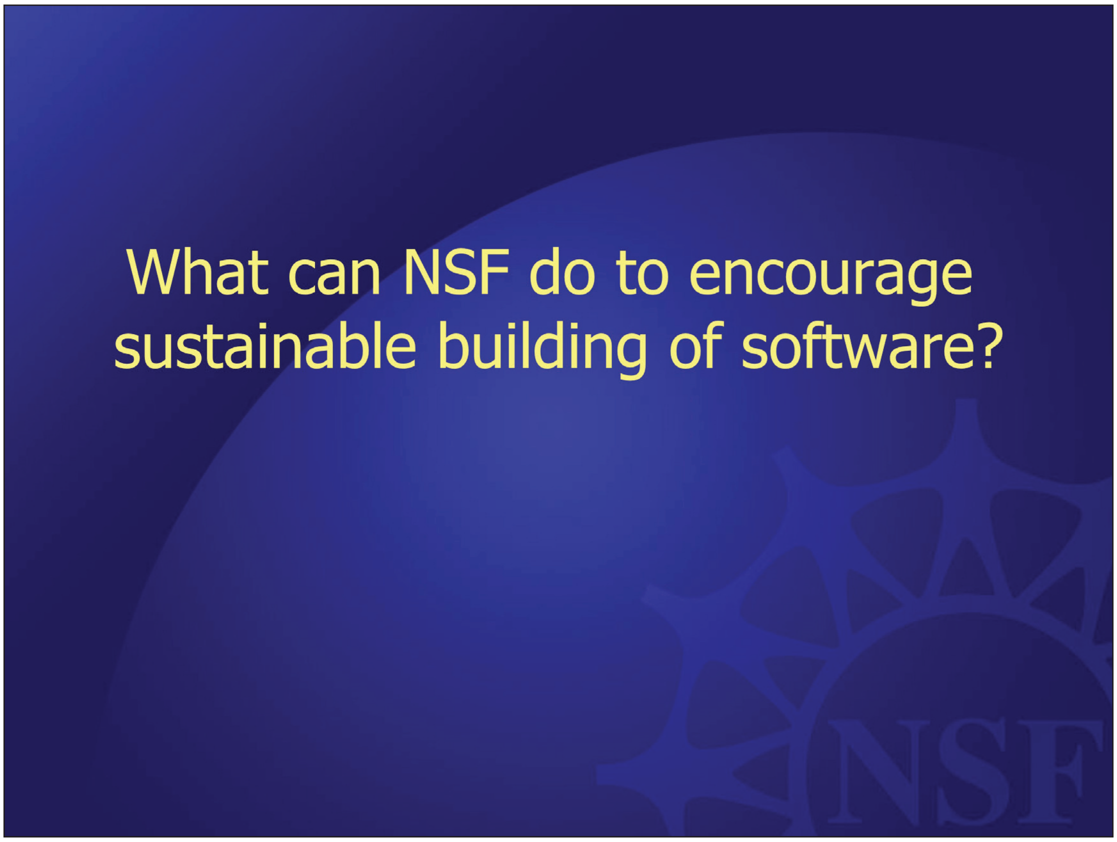
- But how many computational scientists would take this course?

- Carnegie Mellon’s Software Engineering Institute (SEI)
- http://www.sei.cmu.edu/

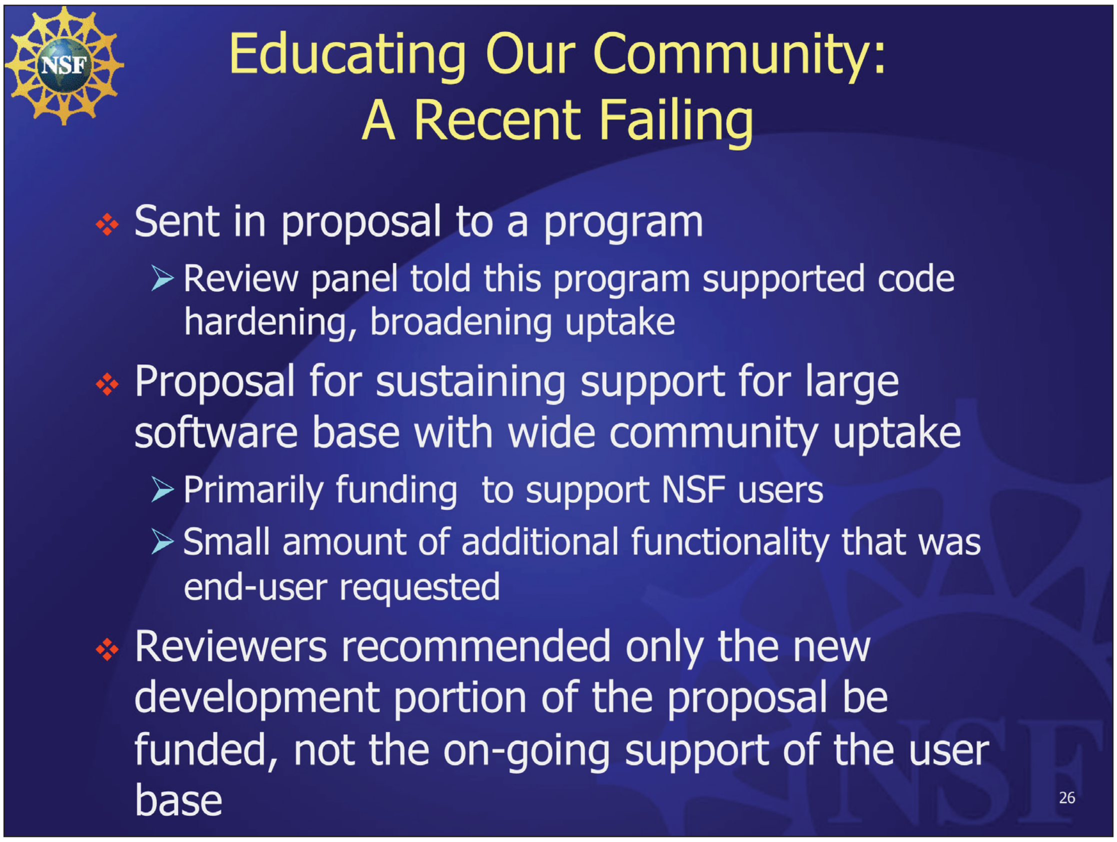
Software Provenance

- Reproducible results are a requirement for basic science
  - In computational science, the science is in the code
- Software must be reliable and consistent
- Version tracking, metadata, environment tracking is critical

- Currently – a vast majority of computational science applications cannot be run by another researcher, and results cannot be reproduced

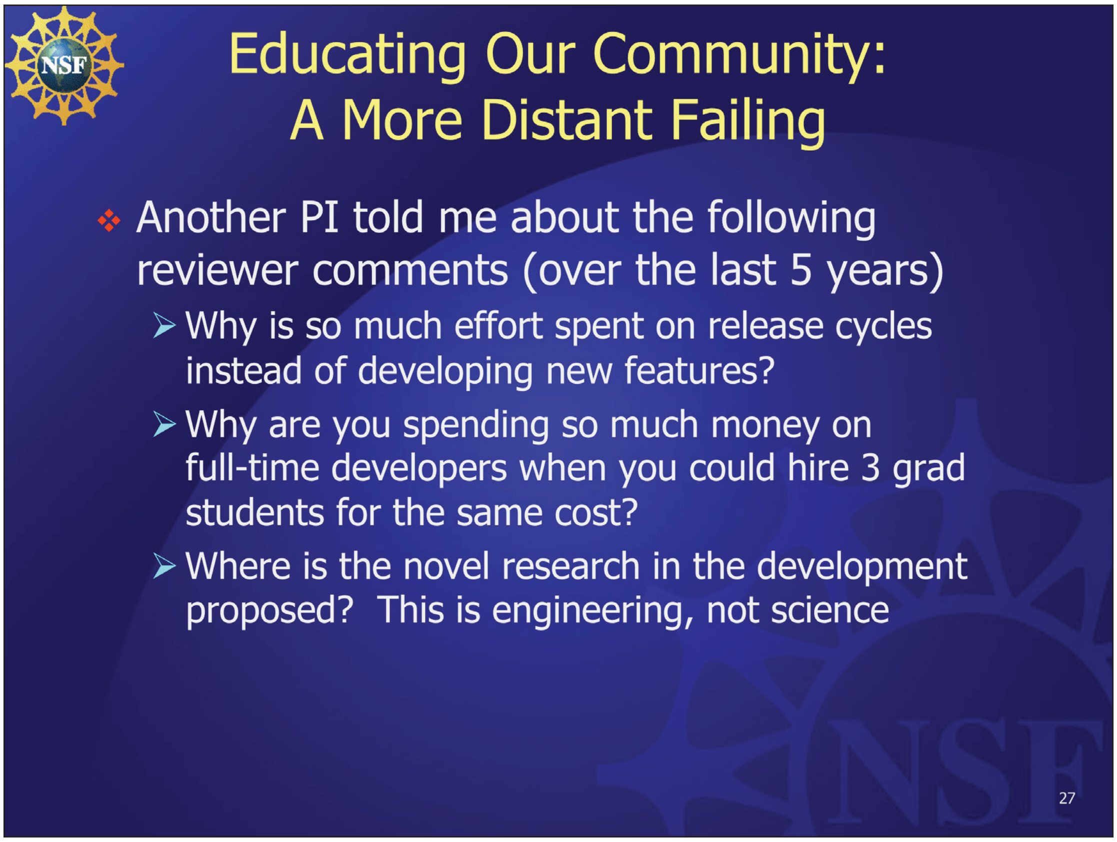


What can NSF do to encourage sustainable building of software?



Educating Our Community: A Recent Failing

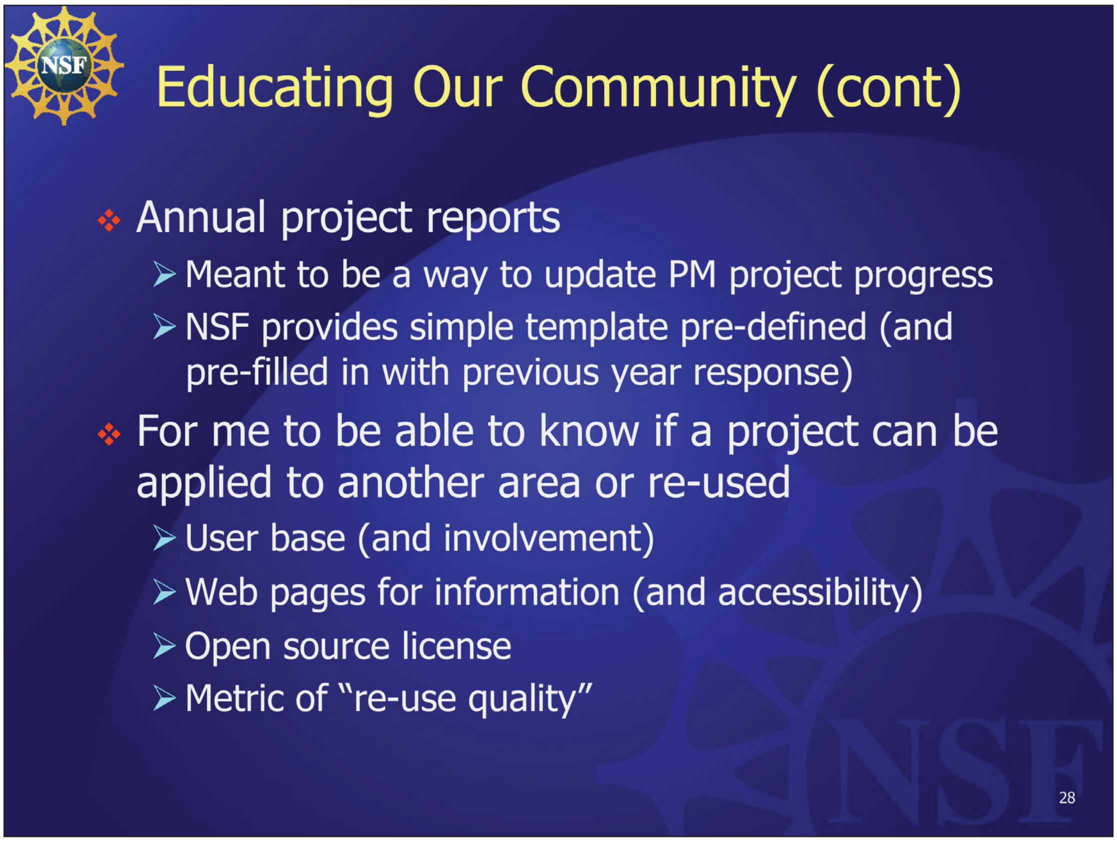
- Sent in proposal to a program
  - Review panel told this program supported code hardening, broadening uptake
- Proposal for sustaining support for large software base with wide community uptake
  - Primarily funding to support NSF users
  - Small amount of additional functionality that was end-user requested
- Reviewers recommended only the new development portion of the proposal be funded, not the on-going support of the user base



Educating Our Community: A More Distant Failing

- Another PI told me about the following reviewer comments (over the last 5 years)
  - Why is so much effort spent on release cycles instead of developing new features?
  - Why are you spending so much money on full-time developers when you could hire 3 grad students for the same cost?
  - Where is the novel research in the development proposed? This is engineering, not science

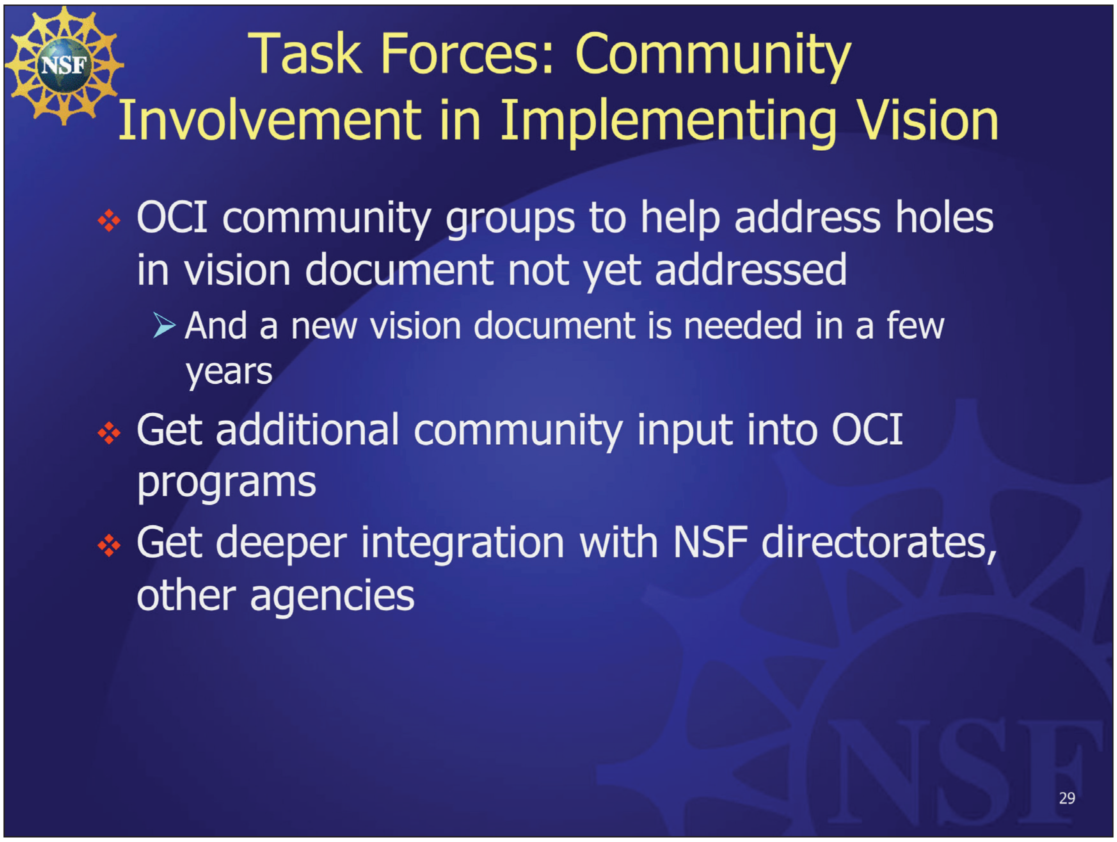
27



Educating Our Community (cont)

- Annual project reports
  - Meant to be a way to update PM project progress
  - NSF provides simple template pre-defined (and pre-filled in with previous year response)
- For me to be able to know if a project can be applied to another area or re-used
  - User base (and involvement)
  - Web pages for information (and accessibility)
  - Open source license
  - Metric of "re-use quality"

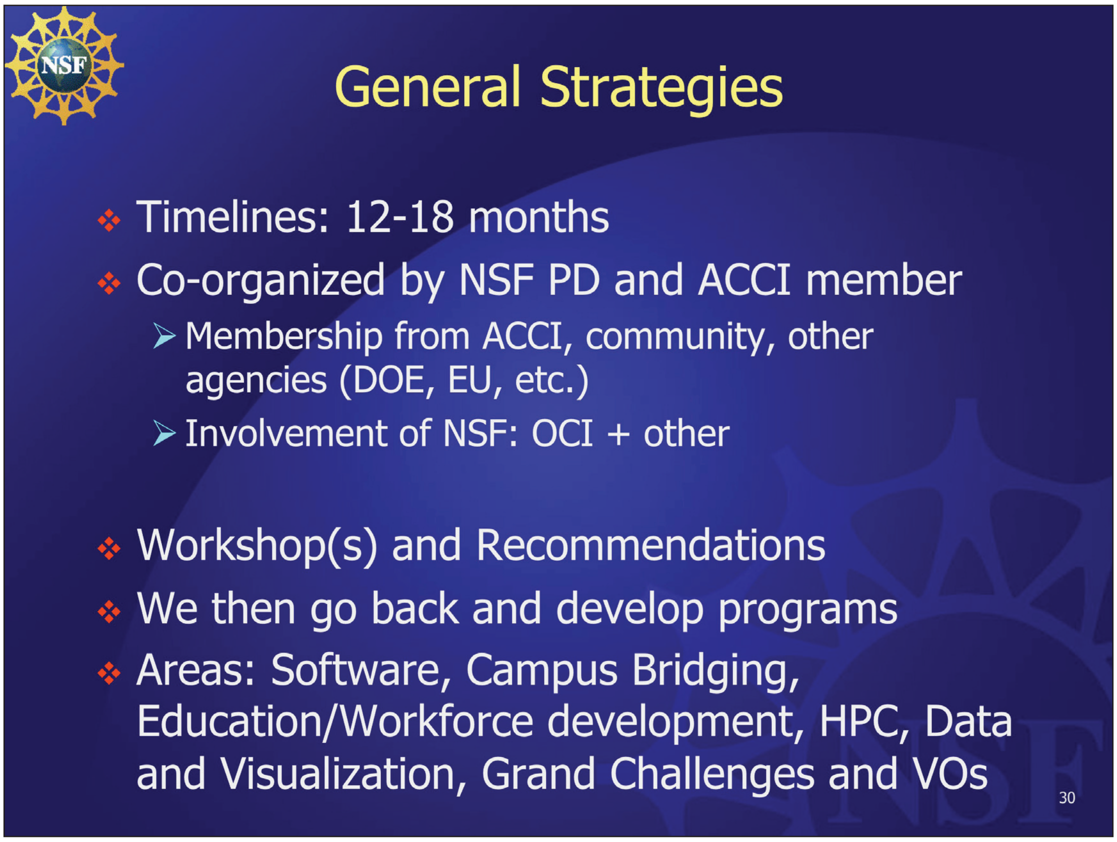
28



Task Forces: Community Involvement in Implementing Vision

- OCI community groups to help address holes in vision document not yet addressed
  - And a new vision document is needed in a few years
- Get additional community input into OCI programs
- Get deeper integration with NSF directorates, other agencies

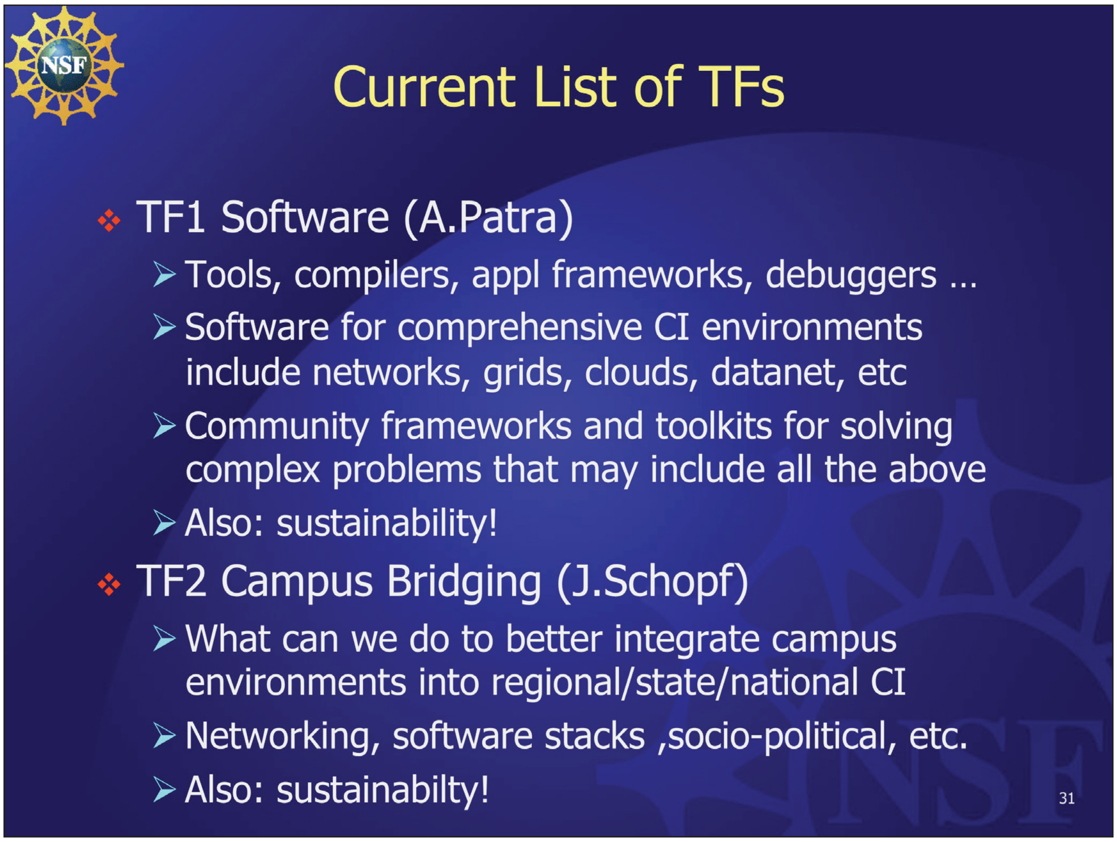
29



General Strategies

- Timelines: 12-18 months
- Co-organized by NSF PD and ACCI member
  - Membership from ACCI, community, other agencies (DOE, EU, etc.)
  - Involvement of NSF: OCI + other
- Workshop(s) and Recommendations
- We then go back and develop programs
- Areas: Software, Campus Bridging, Education/Workforce development, HPC, Data and Visualization, Grand Challenges and VOs

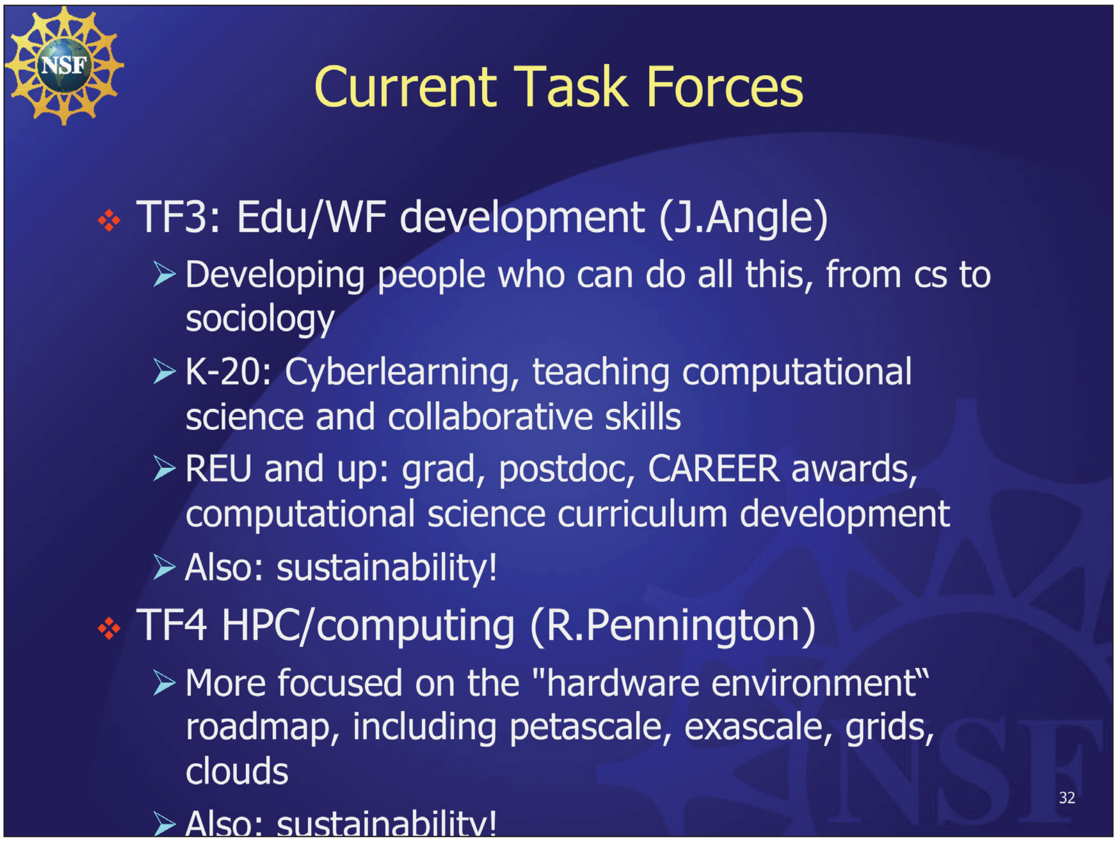
30



Current List of TFs

- TF1 Software (A.Patra)
  - Tools, compilers, appl frameworks, debuggers …
  - Software for comprehensive CI environments include networks, grids, clouds, datanet, etc
  - Community frameworks and toolkits for solving complex problems that may include all the above
  - Also: sustainability!
- TF2 Campus Bridging (J.Schopf)
  - What can we do to better integrate campus environments into regional/state/national CI
  - Networking, software stacks ,socio-political, etc.
  - Also: sustainabilty!

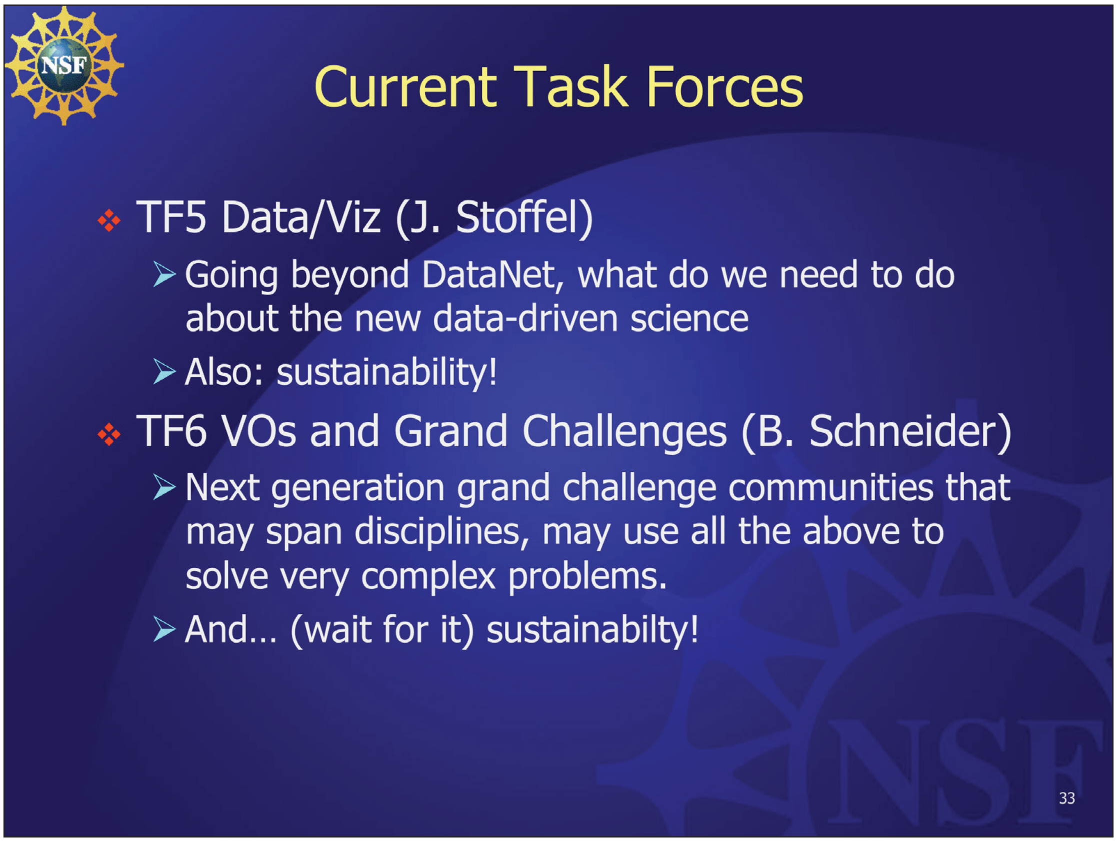
31



Current Task Forces

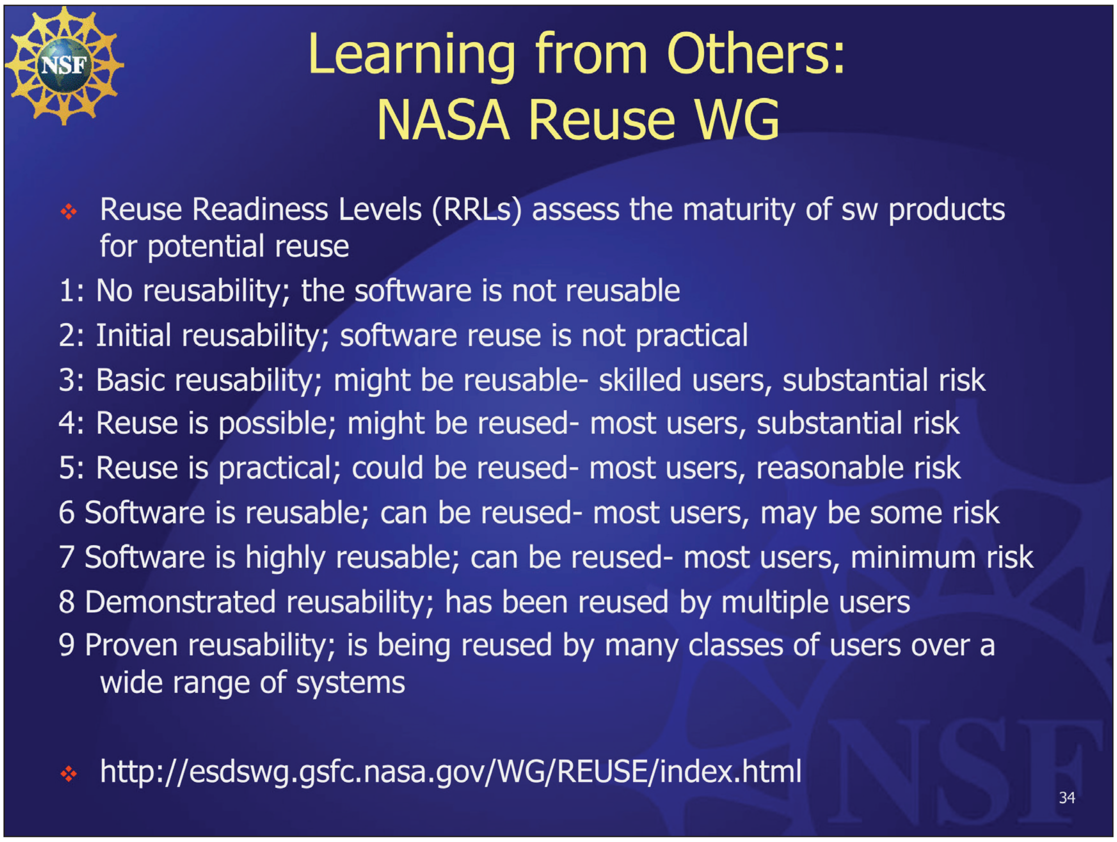
- TF3: Edu/WF development (J.Angle)
  - Developing people who can do all this, from cs to sociology
  - K-20: Cyberlearning, teaching computational science and collaborative skills
  - REU and up: grad, postdoc, CAREER awards, computational science curriculum development
  - Also: sustainability!
- TF4 HPC/computing (R.Pennington)
  - More focused on the "hardware environment" roadmap, including petascale, exascale, grids, clouds
  - Also: sustainability!

32



Current Task Forces

- TF5 Data/Viz (J. Stoffel)
  - Going beyond DataNet, what do we need to do about the new data-driven science
  - Also: sustainability!
- TF6 VOs and Grand Challenges (B. Schneider)
  - Next generation grand challenge communities that may span disciplines, may use all the above to solve very complex problems.
  - And… (wait for it) sustainabilty!

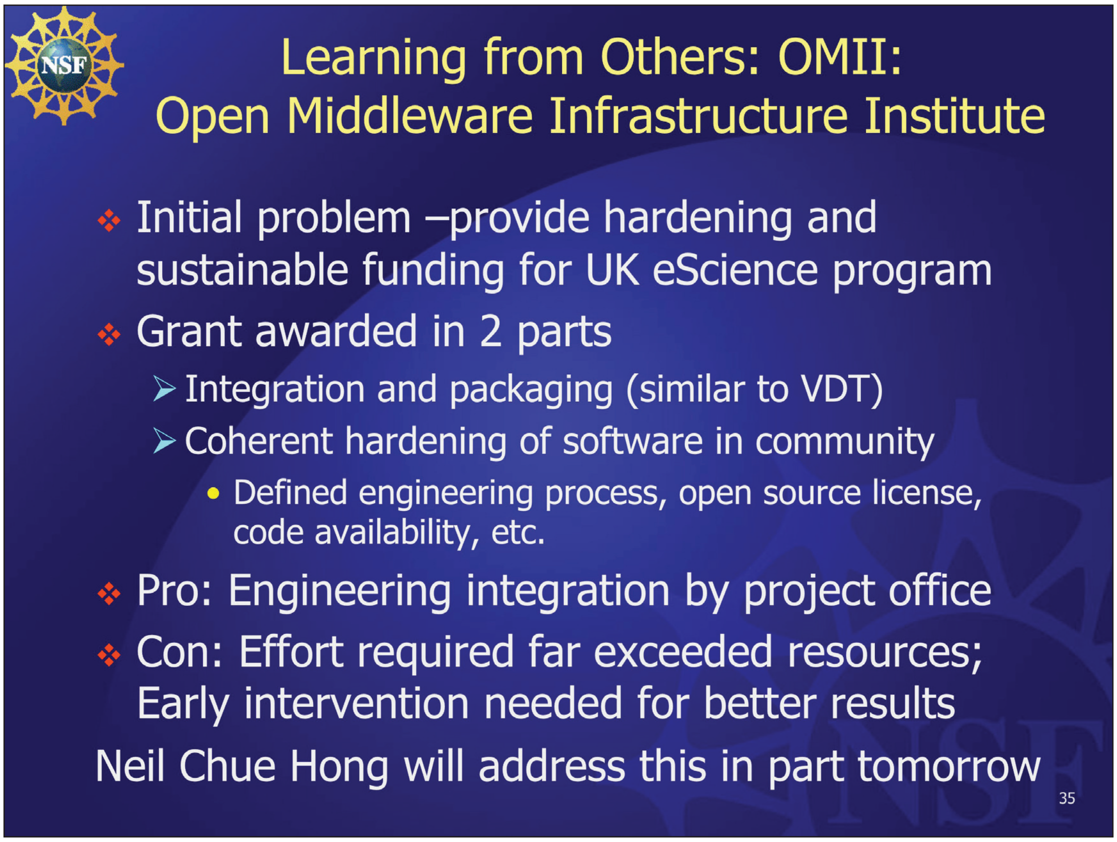


Learning from Others: NASA Reuse WG

- Reuse Readiness Levels (RRLs) assess the maturity of sw products for potential reuse

1: No reusability; the software is not reusable
2: Initial reusability; software reuse is not practical
3: Basic reusability; might be reusable- skilled users, substantial risk
4: Reuse is possible; might be reused- most users, substantial risk
5: Reuse is practical; could be reused- most users, reasonable risk
6 Software is reusable; can be reused- most users, may be some risk
7 Software is highly reusable; can be reused- most users, minimum risk
8 Demonstrated reusability; has been reused by multiple users
9 Proven reusability; is being reused by many classes of users over a wide range of systems

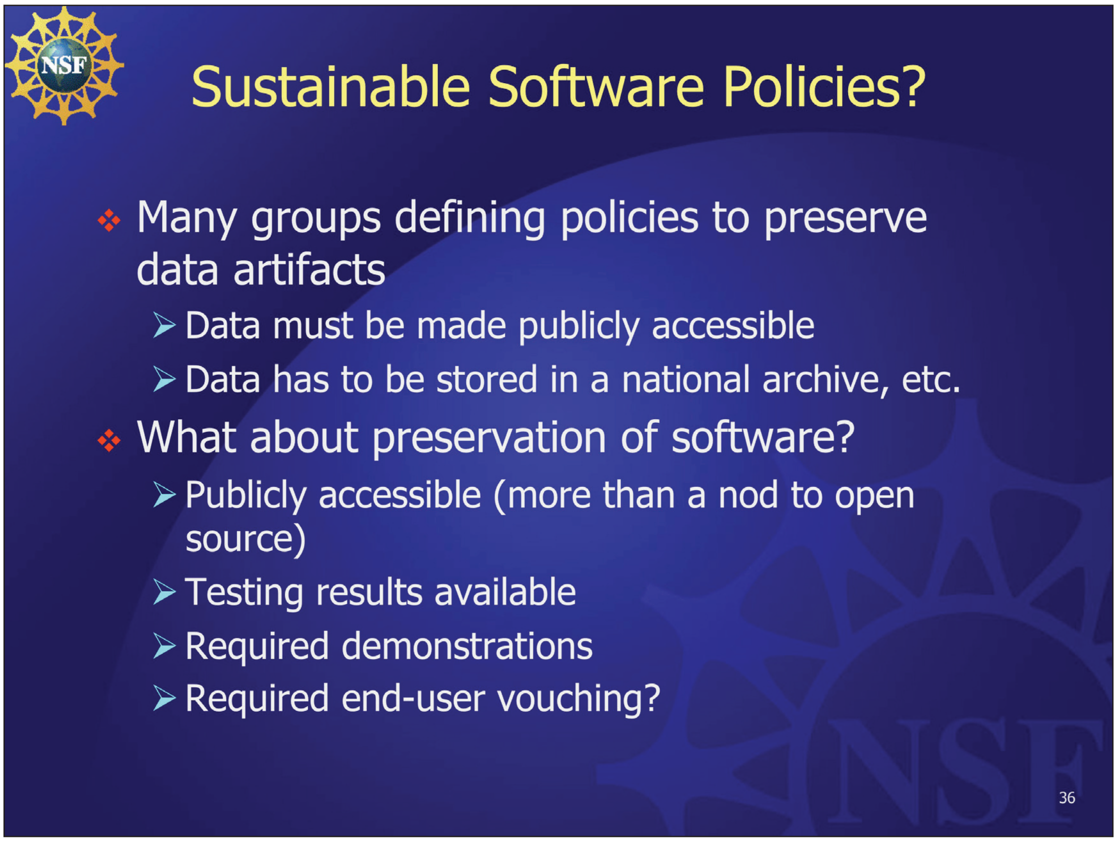
- http://esdswg.gsfc.nasa.gov/WG/REUSE/index.html



Learning from Others: OMII: Open Middleware Infrastructure Institute

- Initial problem –provide hardening and sustainable funding for UK eScience program
- Grant awarded in 2 parts
  - Integration and packaging (similar to VDT)
  - Coherent hardening of software in community
    - Defined engineering process, open source license, code availability, etc.
- Pro: Engineering integration by project office
- Con: Effort required far exceeded resources; Early intervention needed for better results

Neil Chue Hong will address this in part tomorrow



Sustainable Software Policies?

- Many groups defining policies to preserve data artifacts
  - Data must be made publicly accessible
  - Data has to be stored in a national archive, etc.
- What about preservation of software?
  - Publicly accessible (more than a nod to open source)
  - Testing results available
  - Required demonstrations
  - Required end-user vouching?

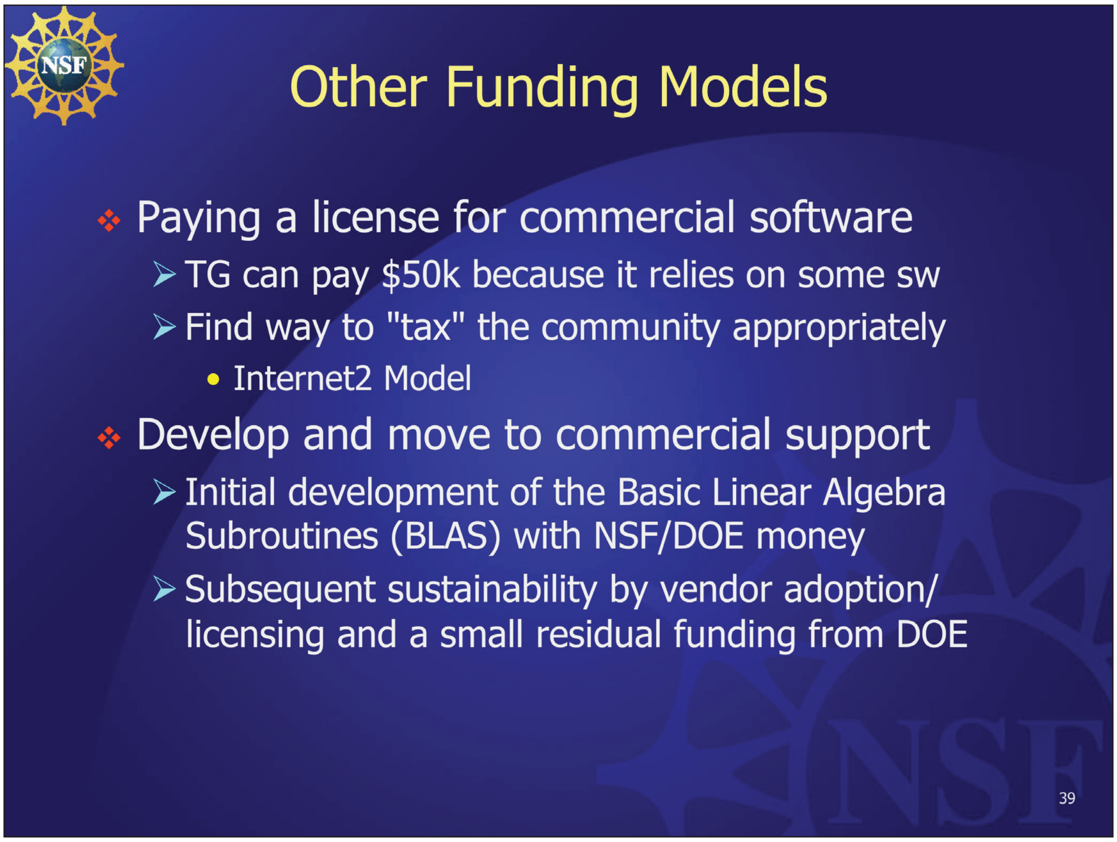
Can we…

- Define metrics of use
- Define metrics of production software not research software
- Capture requirements on the software process (not requirements on functionality itself)
- Make use of professional developers more common place and accepted?

How do we deal with the academic versus production software conflict? How do we reward sustainable software?

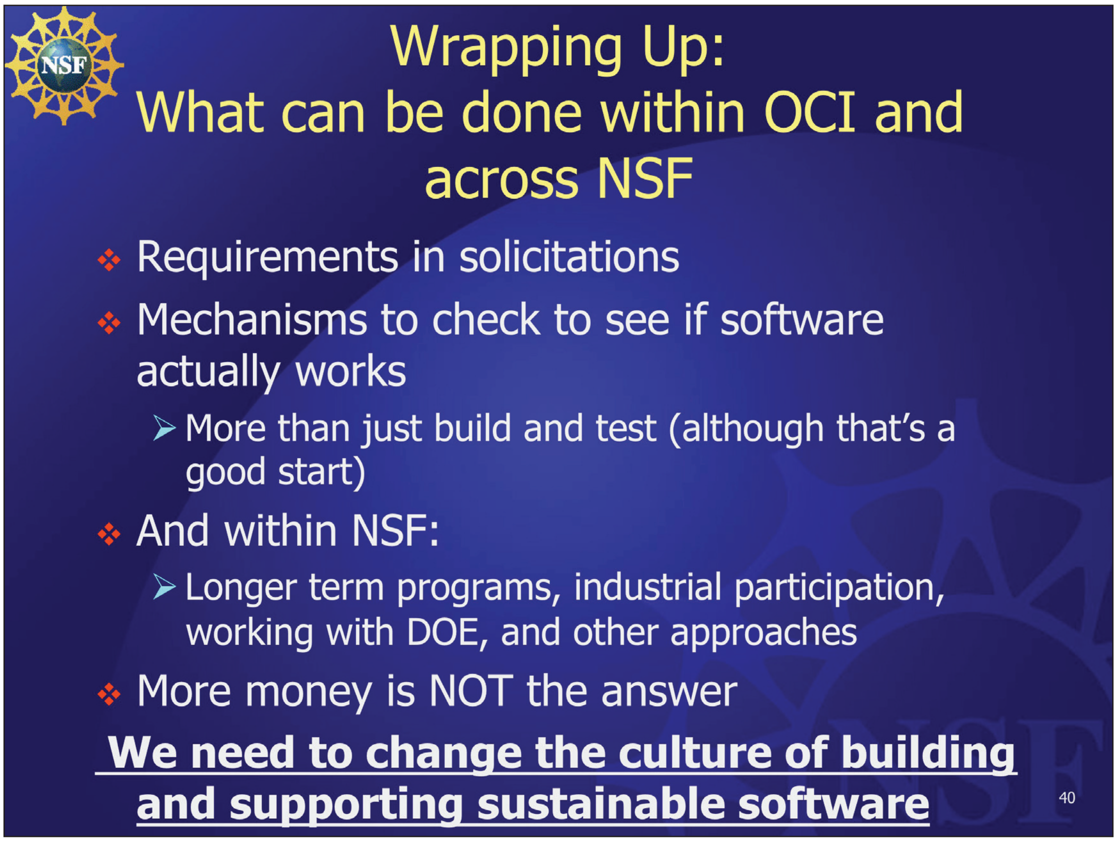
What if we’re successful?

- An effect of successful software reuse is the software stack grows
  - e.g. GSI-OpenSSH on top of OpenSSH on top of GSI on top of OpenSSL
  - Dependencies can cause vulnerabilities and risk



Other Funding Models

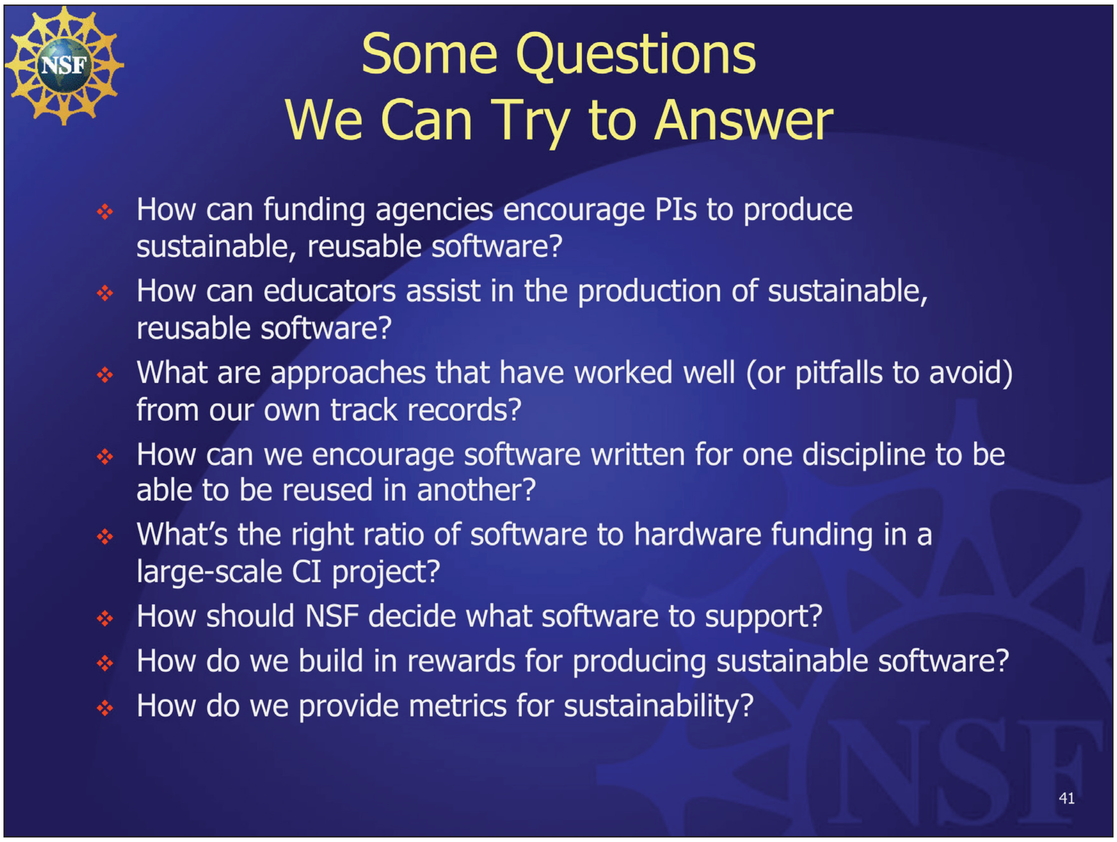
- Paying a license for commercial software
  - TG can pay $50k because it relies on some sw
  - Find way to "tax" the community appropriately
    - Internet2 Model
- Develop and move to commercial support
  - Initial development of the Basic Linear Algebra Subroutines (BLAS) with NSF/DOE money
  - Subsequent sustainability by vendor adoption/ licensing and a small residual funding from DOE



Wrapping Up: What can be done within OCI and across NSF

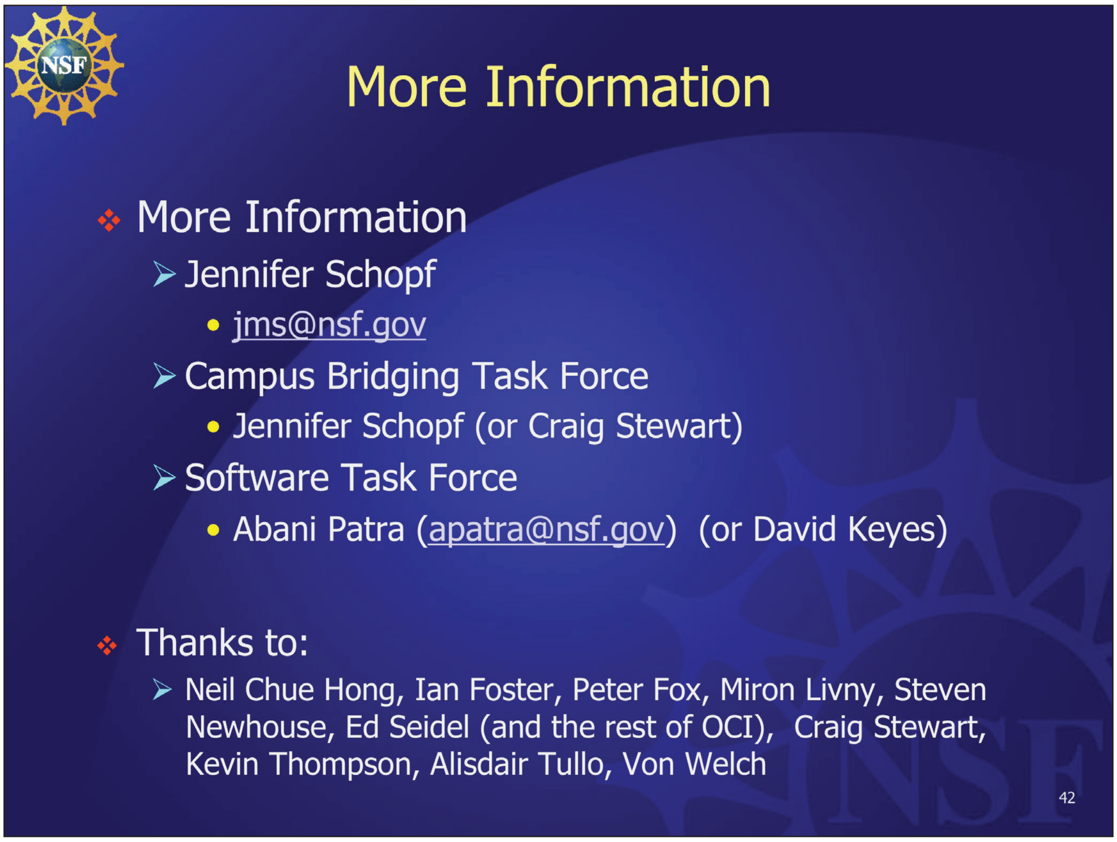
- Requirements in solicitations
- Mechanisms to check to see if software actually works
  - More than just build and test (although that's a good start)
- And within NSF:
  - Longer term programs, industrial participation, working with DOE, and other approaches
- More money is NOT the answer

We need to change the culture of building and supporting sustainable software



Some Questions We Can Try to Answer

- How can funding agencies encourage PIs to produce sustainable, reusable software?
- How can educators assist in the production of sustainable, reusable software?
- What are approaches that have worked well (or pitfalls to avoid) from our own track records?
- How can we encourage software written for one discipline to be able to be reused in another?
- What's the right ratio of software to hardware funding in a large-scale CI project?
- How should NSF decide what software to support?
- How do we build in rewards for producing sustainable software?
- How do we provide metrics for sustainability?



More Information

- More Information
  - Jennifer Schopf
    - jms@nsf.gov
  - Campus Bridging Task Force
    - Jennifer Schopf (or Craig Stewart)
  - Software Task Force
    - Abani Patra (apatra@nsf.gov) (or David Keyes)
- Thanks to:
  - Neil Chue Hong, Ian Foster, Peter Fox, Miron Livny, Steven Newhouse, Ed Seidel (and the rest of OCI), Craig Stewart, Kevin Thompson, Alisdair Tullo, Von Welch

Cultivating Sustainable Software for Research the Perspective from OMII-UK

Cyberinfrastructure Software and Sustainability Workshop

26-27 March 2009

Neil Chue Hong

Director, OMII-UK

omii-uk

Web: www.omii.ac.uk Email: info@omii.ac.uk

Discussion's been "what if"

- What if
  - There had been a large scale cross-funder cyber-infrastructure initiative?
  - There was funding for software development that was separate from research?
  - There was a group set up to take the software outputs of the programme and support continued development?

UK e-Science

e-Science

'e-Science is about global collaboration in key areas of science, and the next generation of infrastructure that will enable it.'

'e-Science will change the dynamic of the way science is undertaken.'

GGF5 Edinburgh

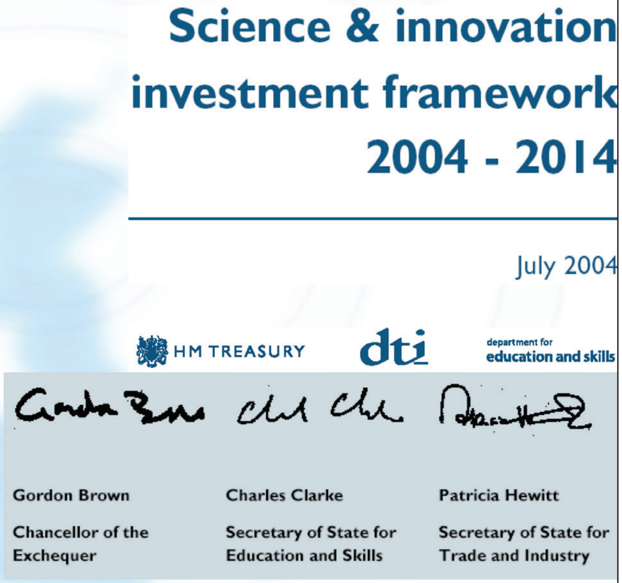
John Taylor
Director General of Research Councils
Office of Science and Technology

From presentation by Tony Hey

e-Science

Investment in e-Infrastructure

- A shared resource
  - That enables science, research, engineering, medicine, industry, …
  - It will improve UK / European / … productivity
    - Lisbon Accord 2000
    - E-Science Vision SR2000 – John Taylor
  - Commitment by UK government
    - Sections 2.23-2.25
  - Always there
    - c.f. telephones, transport, power
  - OSI report
    - www.nesc.ac.uk/documents/OSI/index.html



Science & innovation investment framework 2004 - 2014

July 2004

HM TREASURY dti department for education and skills

[illegible] [illegible] [illegible]

Gordon Brown	Charles Clarke	Patricia Hewitt
Chancellor of the Exchequer	Secretary of State for Education and Skills	Secretary of State for Trade and Industry

e-Science

Embracing Research Diversity

- Thriving Community
  - All disciplines & all Research Councils
  - Industry & Academia
  - Many universities & research institutes
  - Productive collaboration

bbsrc biotechnology and biological sciences research council

JISC

Science & Technology Facilities Council

EPSRC Engineering and Physical Sciences Research Council

E·S·R·C ECONOMIC & SOCIAL RESEARCH COUNCIL

dti

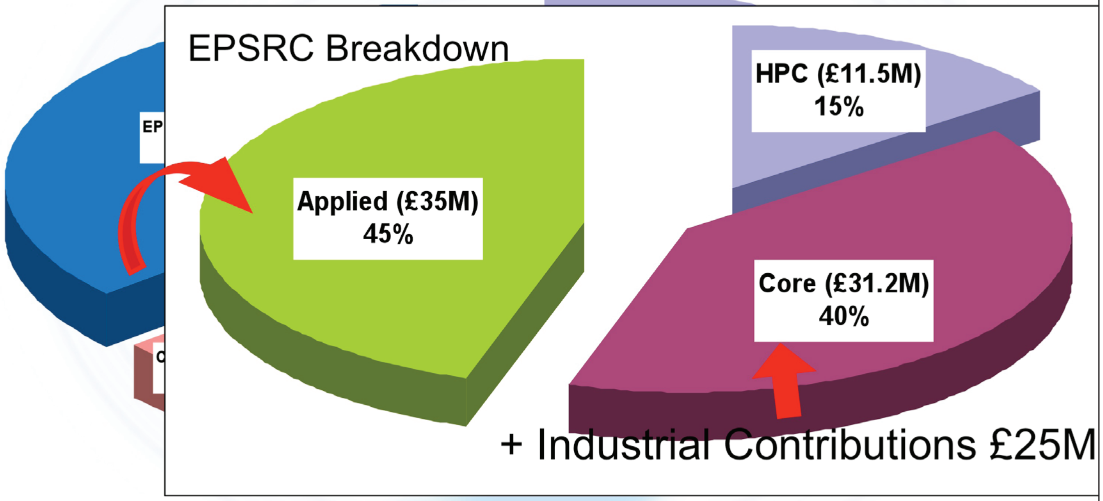
Arts & Humanities Research Council

MRC Medical Research Council

NATURAL ENVIRONMENT RESEARCH COUNCIL

UK e-Science Budget (2001-2006)

Total: £213M + £100M via JISC



Source: Science Budget 2003/4 – 2005/6, DTI(OST)

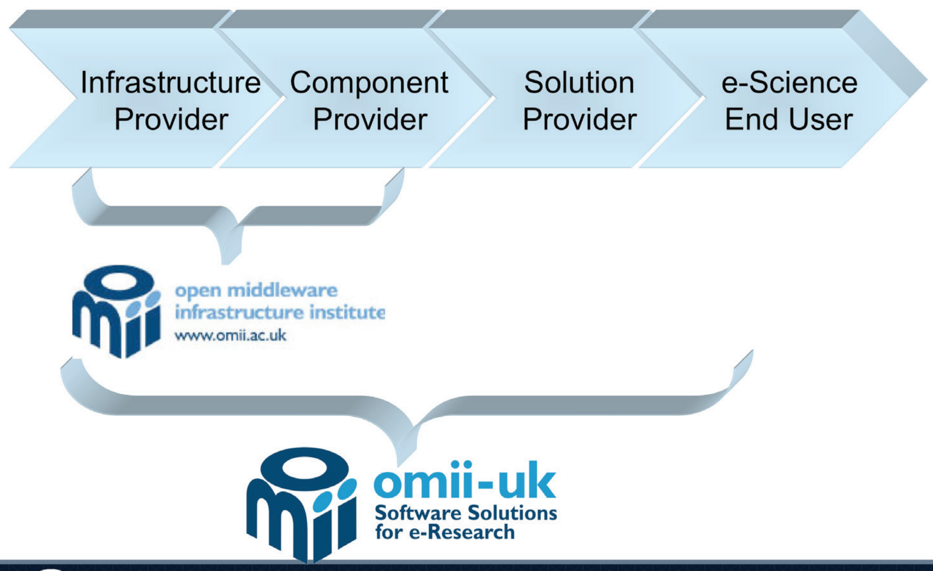
Slide from Steve Newhouse

History

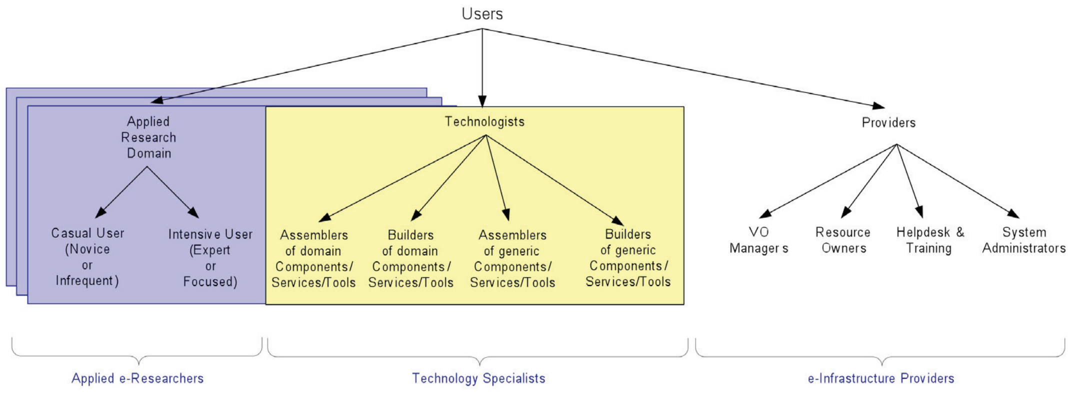
2004 OMII
- IBM-Southampton partnership
- Software engineering and QA process
- Ingest pipeline and managed programme
- Middleware product focused
- OMII WSI-based stack and distribution

2006 OMII UK
- Southampton-Manchester-Edinburgh partnership
- Software engineering and QA process
- Ingest pipeline and contributed programme
- User relevance and adoption focus
- "pick and mix" components
- Collaboration and partnerships

Three "S's": Software, Support, Sustainability



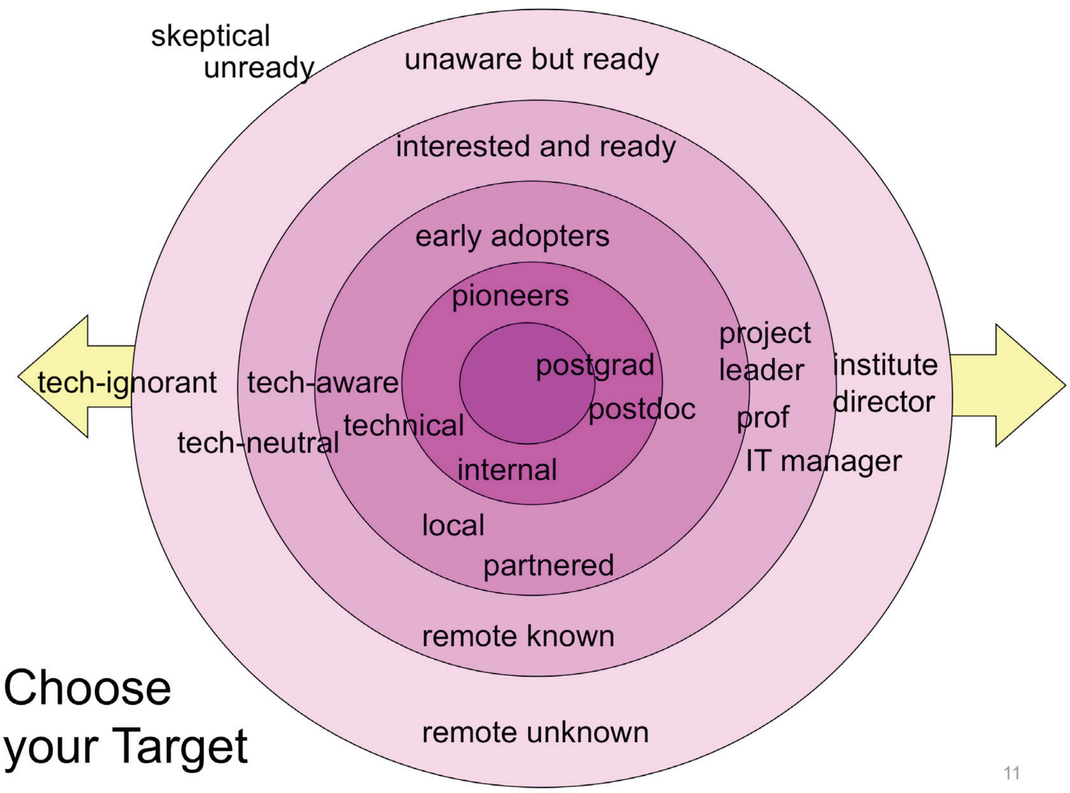
OMII-UK Users


Users
Applied Research Domain
Casual User (Novice or Infrequent)
Intensive User (Expert or Focused)
Technologists
Assemblers of domain Components/ Services/Tools
Builders of domain Components/ Services/Tools
Assemblers of generic Components/ Services/Tools
Builders of generic Components/ Services/Tools
Providers
VO Managers
Resource Owners
Helpdesk & Training
System Administrators
Applied e-Researchers
Technology Specialists
e-Infrastructure Providers

Applied e-Researchers

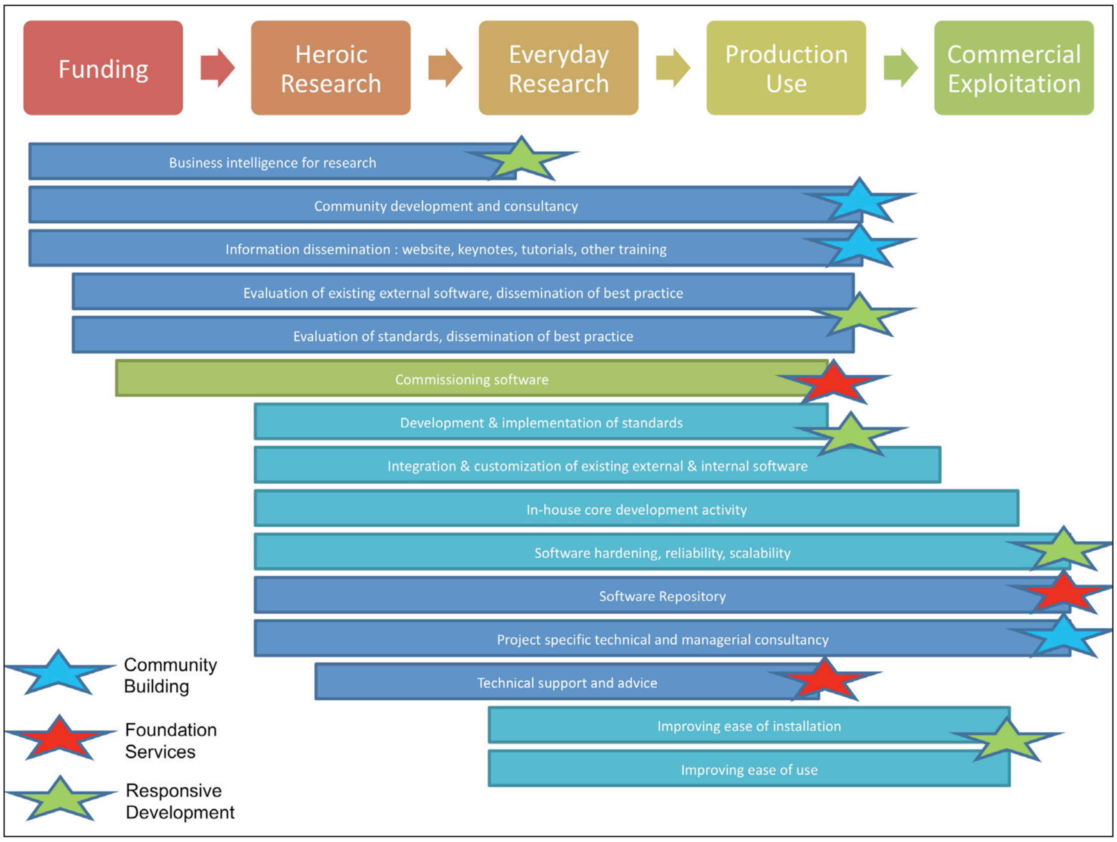
Technology Specialists (domain & generic)

e-Infrastructure Providers

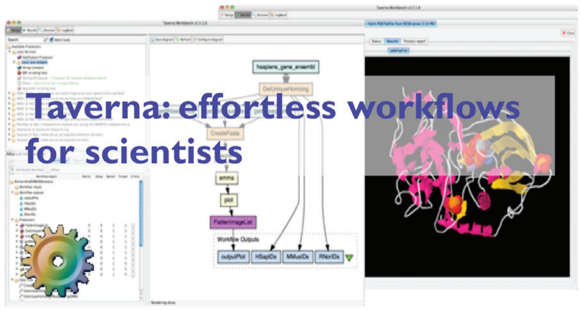


Choose your Target

11

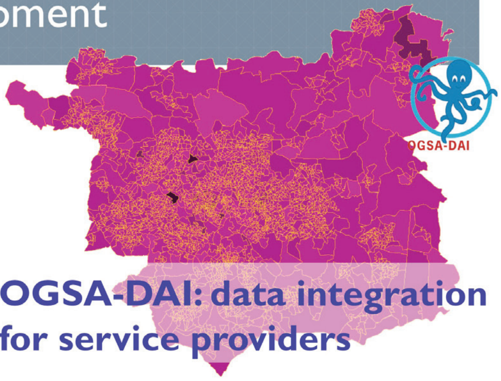

Funding
Heroic Research
Everyday Research
Production Use
Commercial Exploitation
Business intelligence for research
Community development and consultancy
Information dissemination : website, keynotes, tutorials, other training
Evaluation of existing external software, dissemination of best practice
Evaluation of standards, dissemination of best practice
Commissioning software
Development & implementation of standards
Integration & customization of existing external & internal software
In-house core development activity
Software hardening, reliability, scalability
Software Repository
Project specific technical and managerial consultancy
Technical support and advice
Improving ease of installation
Improving ease of use
Community Building
Foundation Services
Responsive Development

OMII-UK: Software Development

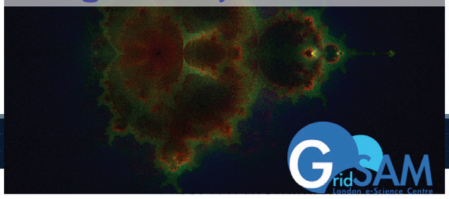


Taverna: effortless workflows for scientists

Taverna user Interface with a workflow diagram and the result panel displaying a protein structure.



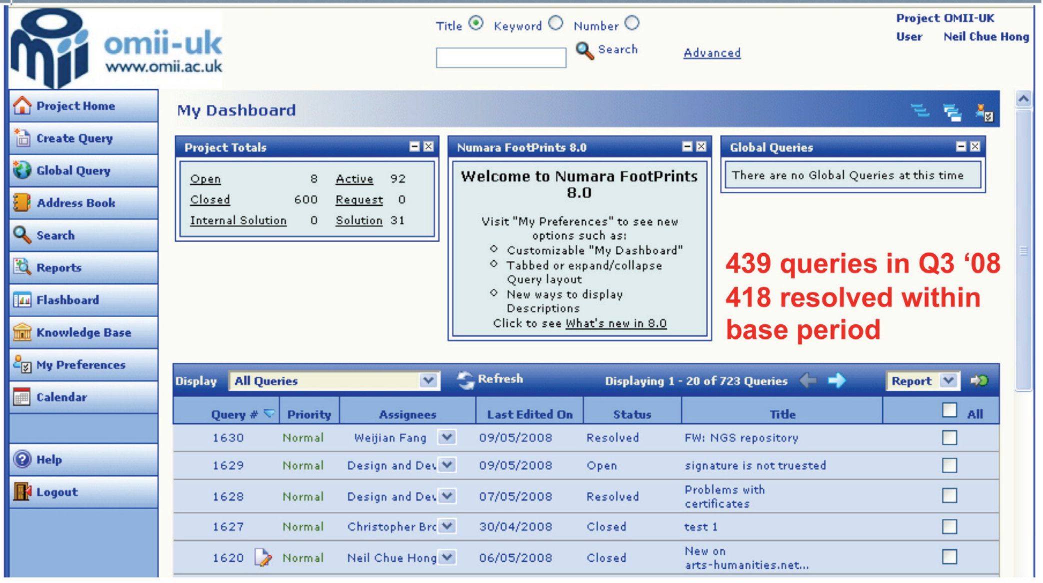
OGSA-DAI: data integration for service providers



Campus Grid Toolkit: easy to install grid for job submission

PAG: AG videoconferencing for anyone

Support and Helpdesk



omii-uk www.omii.ac.uk

Title [✓] Keyword Number Search Advanced

Project OMII-UK
User Neil Chue Hong

Project Home · Create Query · Global Query · Address Book · Search · Reports · Flashboard · Knowledge Base · My Preferences · Calendar · Help · Logout

My Dashboard

Project Totals

Open	8	Active	92
Closed	600	Request	0
Internal Solution	0	Solution	31

Numara FootPrints 8.0

Welcome to Numara FootPrints 8.0

Visit "My Preferences" to see new options such as:
- Customizable "My Dashboard"
- Tabbed or expand/collapse Query layout
- New ways to display Descriptions

Click to see What's new in 8.0

Global Queries

There are no Global Queries at this time

439 queries in Q3 '08
418 resolved within base period

Display All Queries · Refresh · Displaying 1 - 20 of 723 Queries · Report

Query #	Priority	Assignees	Last Edited On	Status	Title	All
1630	Normal	Weijian Fang	09/05/2008	Resolved	FW: NGS repository	☐
1629	Normal	Design and Dev	09/05/2008	Open	signature is not truested	☐
1628	Normal	Design and Dev	07/05/2008	Resolved	Problems with certificates	☐
1627	Normal	Christopher Brc	30/04/2008	Closed	test 1	☐
1620	Normal	Neil Chue Hong	06/05/2008	Closed	New on arts-humanities.net...	☐

omii-uk

Web: www.omii.ac.uk Email: info@omii.ac.uk

Engaging Research with e-Infrastructure

engage
Engaging research with e-Infrastructure

- 53 interviews
  - 36 face-to-face
  - 17 telephone
- 60 people
- 24 institutions

- Triage process to identify development projects and best practice

What's the issue?

What's the issue?

"Sustainability is not an issue for CS researchers, we want others to take the software over subject to IP issues"

What's the issue?

"Sustainability is a big issue, we are producing complex tools we want to continue to use but it's not clear how they'll be sustained"

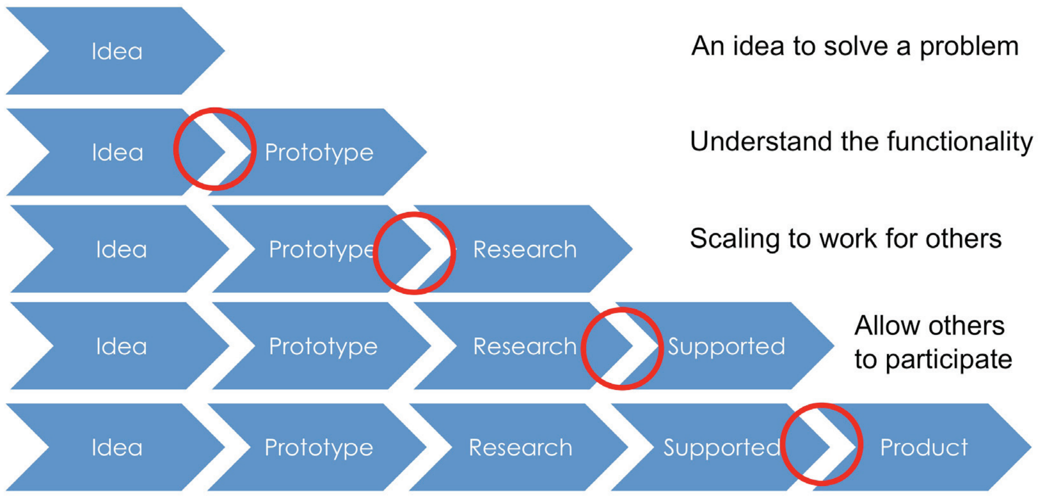
What's the issue?

- How do we ensure that software which has users can continue to support them?
- In particular how do we help software survive the transition from creation to through adoption to widespread use and beyond?
- Some examples from OMII-UK experiences

Software development comes in stages

(and it takes time)

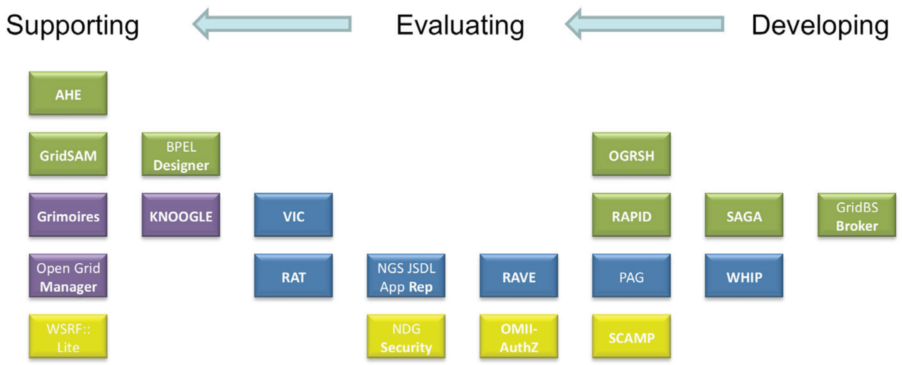
Software development comes in stages


Idea
An idea to solve a problem
Idea
Prototype
Understand the functionality
Idea
Prototype
Research
Scaling to work for others
Idea
Prototype
Research
Supported
Allow others to participate
Idea
Prototype
Research
Supported
Product

Commissioned Software Programme

- Identify gaps in functionality or quality required from community
- Scaling from one user to a small group of users, one set of developers to many developers
- Criteria for judging CSP development:
  - the demand for the software is understood
  - the number of potential users has been increased by the work done
  - the use of the software has contributed to a measurable increase in the research outputs
  - the community participation around the software has increased

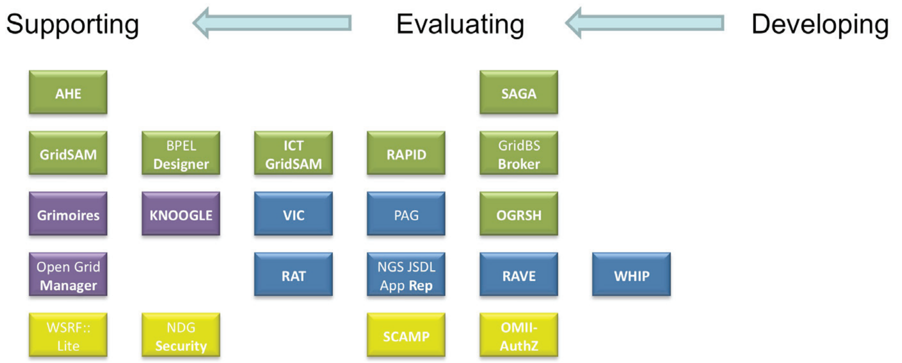
Commissioned Software Programme @ Q1 '08



omii-uk

Web: www.omii.ac.uk Email: info@omii.ac.uk

Commissioned Software Programme @ Q3 '08

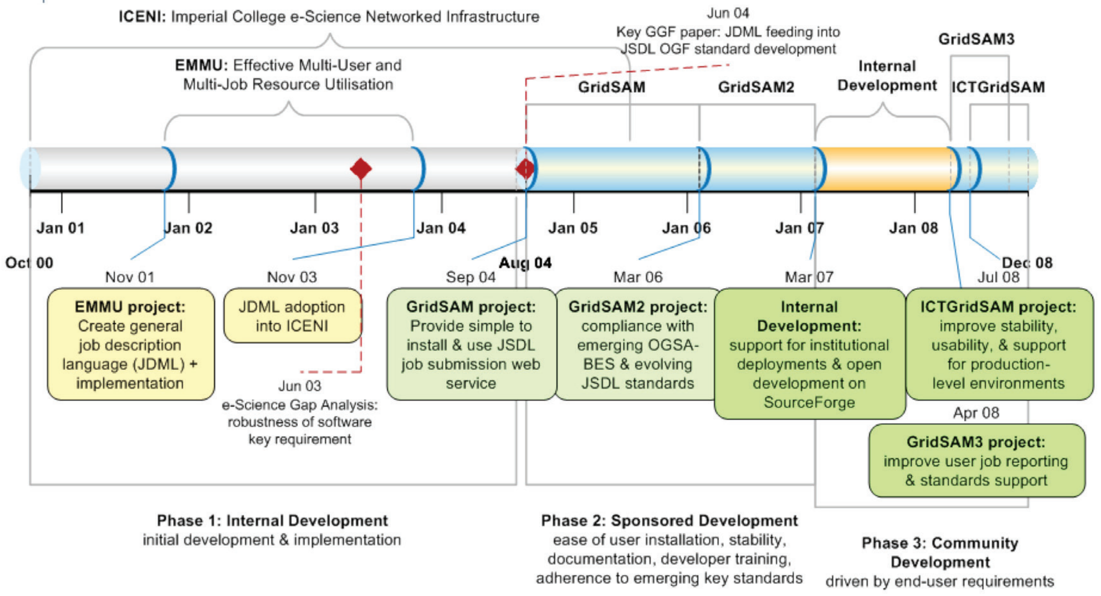


Commissioned Software Projects progress through the software lifecycle

omii-uk

Web: www.omii.ac.uk Email: info@omii.ac.uk

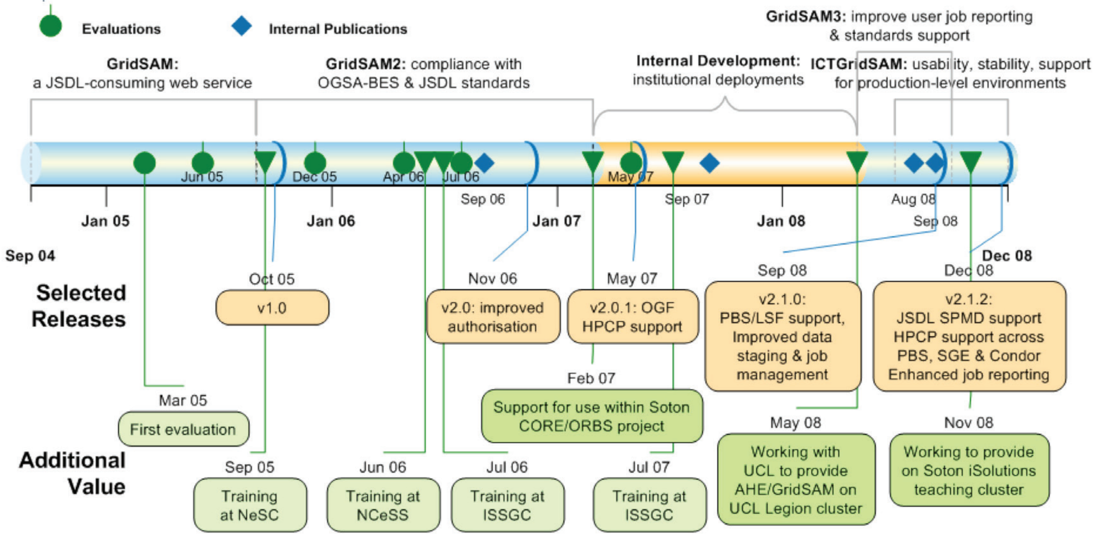
GridSAM – History & OMII-UK Involvement


ICENI: Imperial College e-Science Networked Infrastructure
EMMU: Effective Multi-User and Multi-Job Resource Utilisation
Jun 04
Key GGF paper: JDML feeding into JSDL OGF standard development
GridSAM
GridSAM2
Internal Development
GridSAM3
ICTGridSAM
Jan 01
Jan 02
Jan 03
Jan 04
Jan 05
Jan 06
Jan 07
Jan 08
Oct 00
Aug 04
Dec 08
Nov 01
Nov 03
Sep 04
Mar 06
Mar 07
Jul 08
EMMU project: Create general job description language (JDML) + implementation
JDML adoption into ICENI
GridSAM project: Provide simple to install & use JSDL job submission web service
GridSAM2 project: compliance with emerging OGSA-BES & evolving JSDL standards
Internal Development: support for institutional deployments & open development on SourceForge
ICTGridSAM project: improve stability, usability, & support for production-level environments
Jun 03
e-Science Gap Analysis: robustness of software key requirement
Apr 08
GridSAM3 project: improve user job reporting & standards support
Phase 1: Internal Development
initial development & implementation
Phase 2: Sponsored Development
ease of user installation, stability, documentation, developer training, adherence to emerging key standards
Phase 3: Community Development
driven by end-user requirements

omii-uk

Web: www.omii.ac.uk Email: info@omii.ac.uk

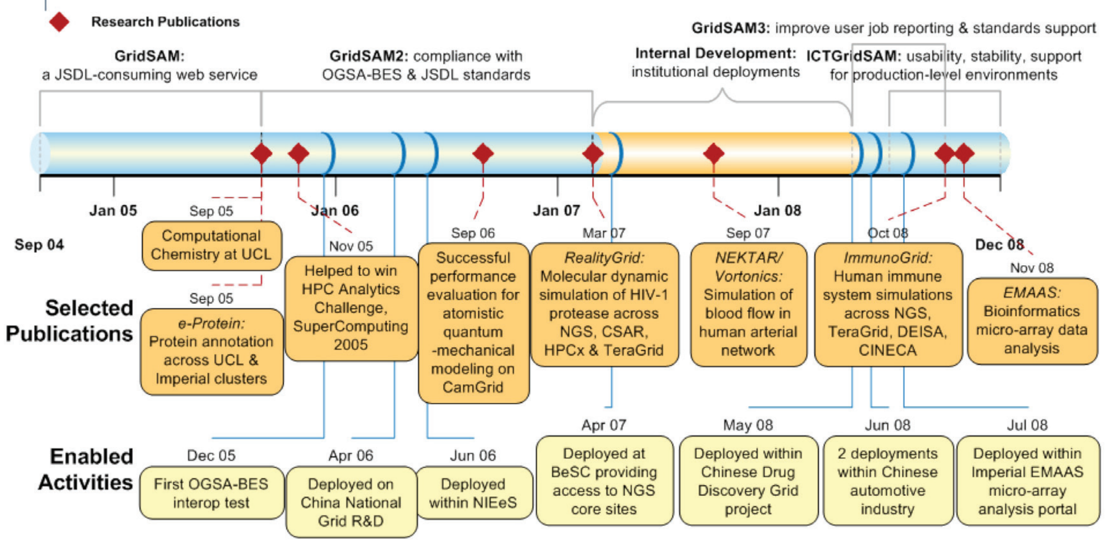
GridSAM – Releases & Additional Value


● Evaluations
◆ Internal Publications
GridSAM3: improve user job reporting & standards support
GridSAM: a JSDL-consuming web service
GridSAM2: compliance with OGSA-BES & JSDL standards
Internal Development: institutional deployments
ICTGridSAM: usability, stability, support for production-level environments
Jun 05
Dec 05
Apr 06
Jul 06
May 07
Sep 06
Sep 07
Aug 08
Sep 08
Jan 05
Jan 06
Jan 07
Jan 08
Sep 04
Dec 08
Selected Releases
Oct 05
v1.0
Nov 06
v2.0: improved authorisation
May 07
v2.0.1: OGF HPCP support
Sep 08
v2.1.0: PBS/LSF support, Improved data staging & job management
Dec 08
v2.1.2: JSDL SPMD support HPCP support across PBS, SGE & Condor Enhanced job reporting
Feb 07
Support for use within Soton CORE/ORBS project
Mar 05
First evaluation
Additional Value
Sep 05
Training at NeSC
Jun 06
Training at NCeSS
Jul 06
Training at ISSGC
Jul 07
Training at ISSGC
May 08
Working with UCL to provide AHE/GridSAM on UCL Legion cluster
Nov 08
Working to provide on Soton iSolutions teaching cluster

omii-uk

Web: www.omii.ac.uk Email: info@omii.ac.uk

GridSAM – Publications & Enabled Activities



Research Publications
GridSAM: a JSDL-consuming web service
GridSAM2: compliance with OGSA-BES & JSDL standards
Internal Development: institutional deployments
GridSAM3: improve user job reporting & standards support
ICTGridSAM: usability, stability, support for production-level environments
Sep 04
Jan 05
Jan 06
Jan 07
Jan 08
Dec 08
Selected Publications
Sep 05
Computational Chemistry at UCL
Sep 05
e-Protein: Protein annotation across UCL & Imperial clusters
Nov 05
Helped to win HPC Analytics Challenge, SuperComputing 2005
Sep 06
Successful performance evaluation for atomistic quantum -mechanical modeling on CamGrid
Mar 07
RealityGrid: Molecular dynamic simulation of HIV-1 protease across NGS, CSAR, HPCx & TeraGrid
Sep 07
NEKTAR/ Vortonics: Simulation of blood flow in human arterial network
Oct 08
ImmunoGrid: Human immune system simulations across NGS, TeraGrid, DEISA, CINECA
Nov 08
EMAAS: Bioinformatics micro-array data analysis
Enabled Activities
Dec 05
First OGSA-BES interop test
Apr 06
Deployed on China National Grid R&D
Jun 06
Deployed within NIEeS
Apr 07
Deployed at BeSC providing access to NGS core sites
May 08
Deployed within Chinese Drug Discovery Grid project
Jun 08
2 deployments within Chinese automotive industry
Jul 08
Deployed within Imperial EMAAS micro-array analysis portal

OMII-UK on the Campus

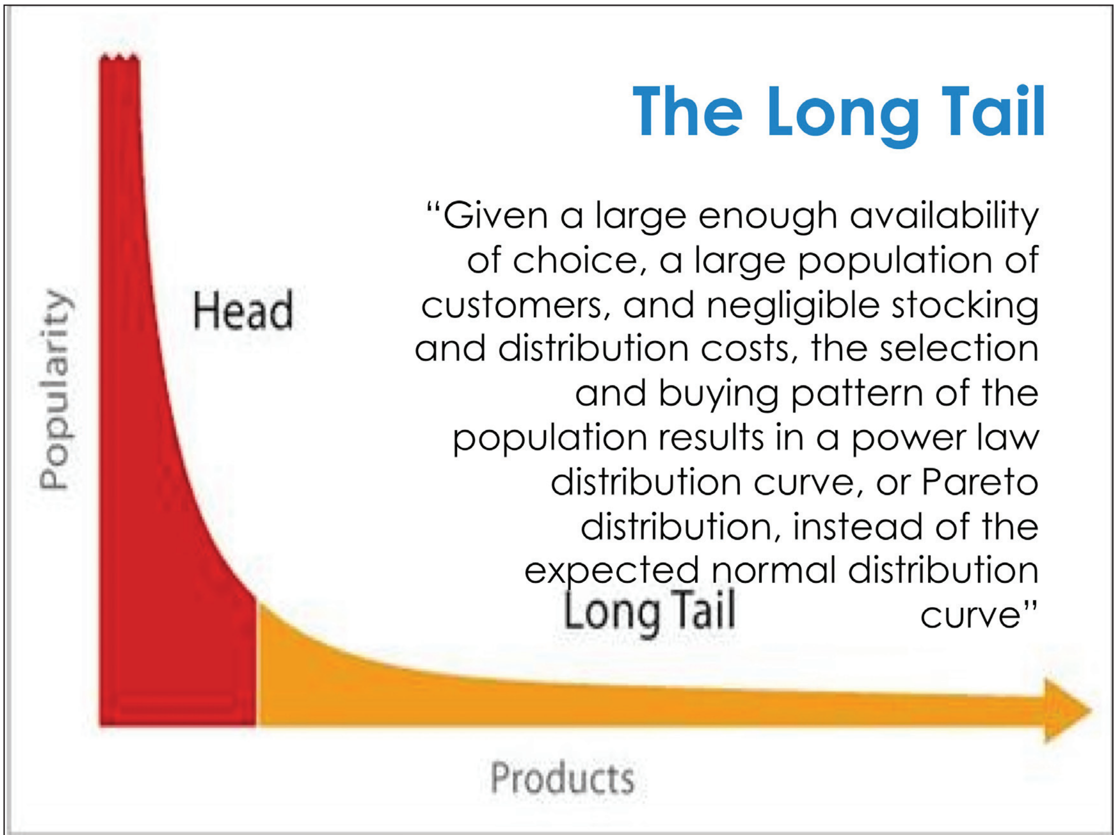
- Rolling out AHE/Campus Grid Toolkit to UCL Research Computing users
  - rolling in contributions from IC, ICT and Oxford
- Looking to repeat this at NWGrid, University of Edinburgh (ECDF) and Kings College London

Linking and Querying of Ancient Texts

- Investigating commentary on inscriptions and papyri
- Develop framework for linking up diverse data sources
- *Gabriel Bodard, Charlotte Rouche*
- **4 different databases now linked as joins rather than unions allowing new epigraphic approaches**

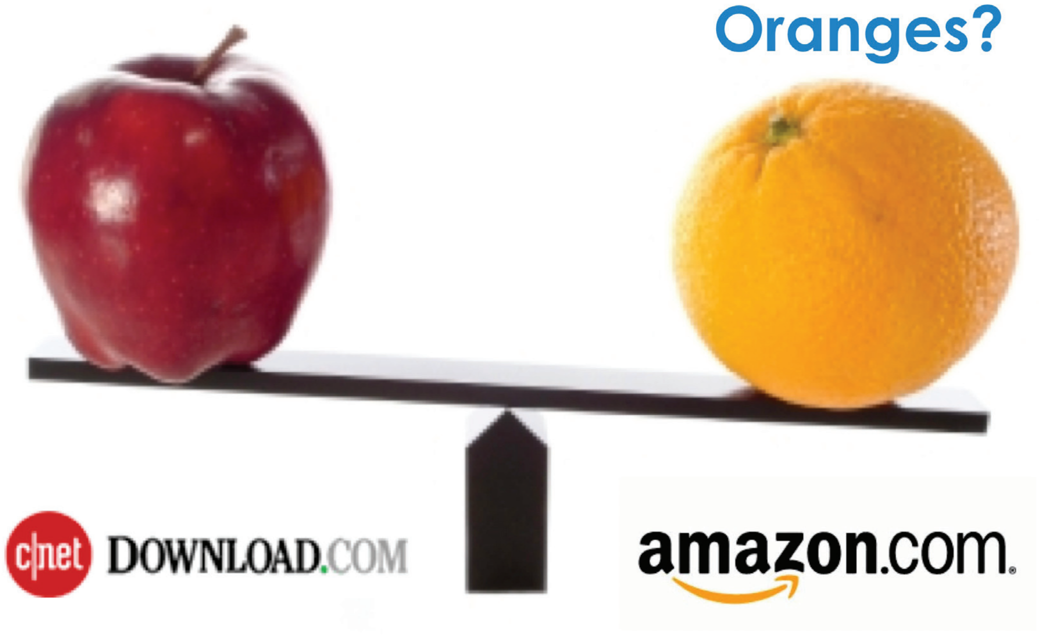
There isn't a single best model for sustainability

The Long Tail



"Given a large enough availability of choice, a large population of customers, and negligible stocking and distribution costs, the selection and buying pattern of the population results in a power law distribution curve, or Pareto distribution, instead of the expected normal distribution curve"

Comparing Apples and Oranges?



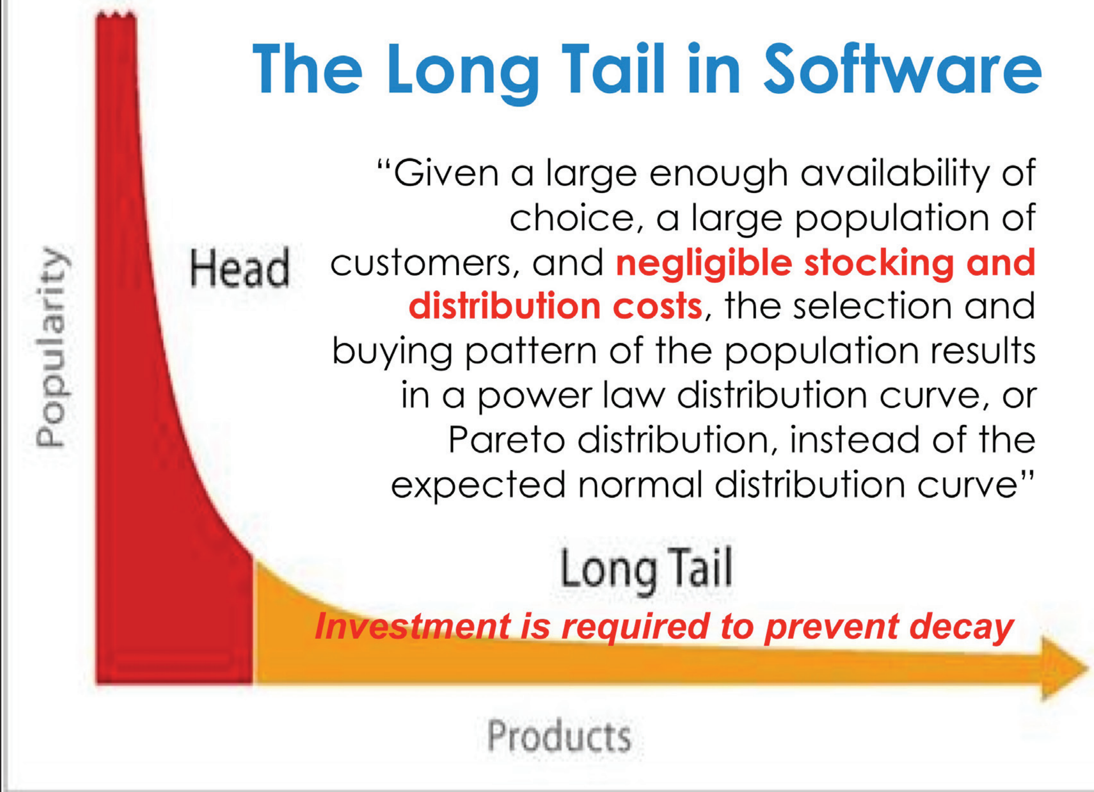
Comparing Apples and Books!

Storage lifespan:
~12 months

Storage lifespan:
~ 50 years

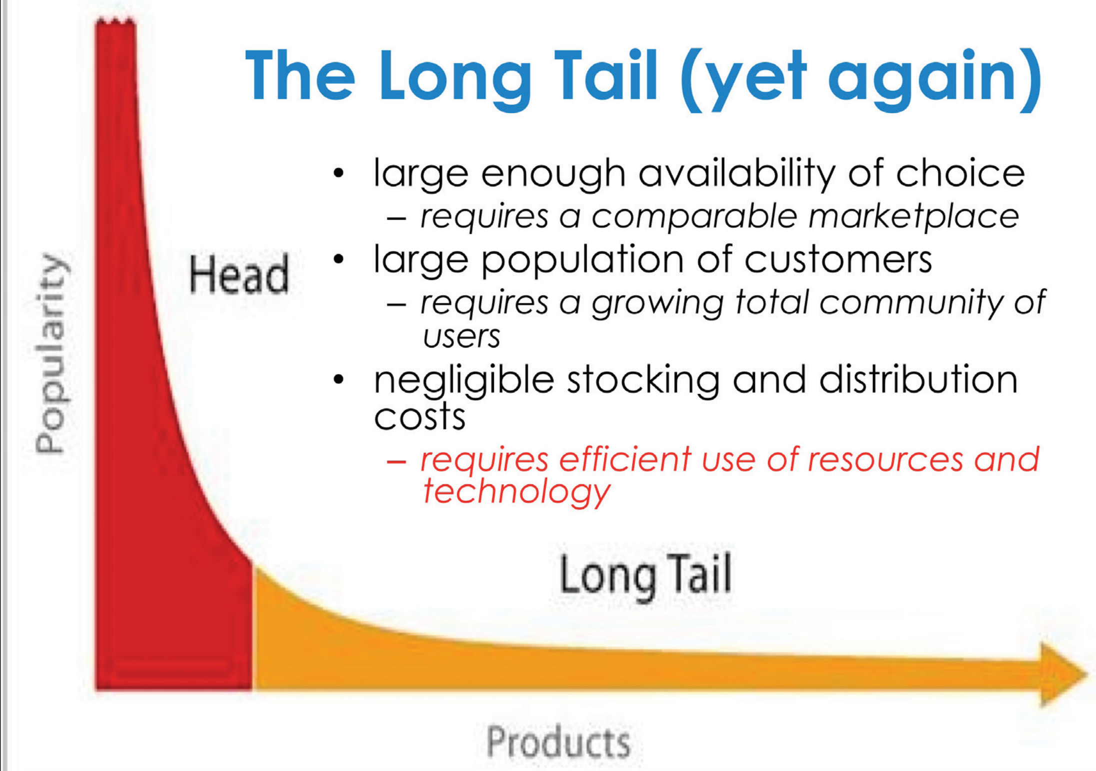
Which one is closer to the lifespan of software?

The Long Tail in Software



"Given a large enough availability of choice, a large population of customers, and **negligible stocking and distribution costs**, the selection and buying pattern of the population results in a power law distribution curve, or Pareto distribution, instead of the expected normal distribution curve"

Investment is required to prevent decay



The Long Tail (yet again)

- large enough availability of choice
  - *requires a comparable marketplace*
- large population of customers
  - *requires a growing total community of users*
- negligible stocking and distribution costs
  - *requires efficient use of resources and technology*

Open Source software is free…

Free as in speech…

free as in beer, or…

Free as in Puppy…

- Long term costs
- Needs love and attention
- May lose charm after growing up
- Occasional clean-ups required
- Many left abandoned by their owners

How to embed e-Infrastructure in the research process?

- What do you think e-Infrastructure is and what should it be?
- To what extent is the use of e-Infrastructure essential to your research?
- How would you use e-Infrastructure in the future?
- Do researchers need a clearly defined ICT environment and tool suite?
- What would be needed to truly embed the use of e-Infrastructure in your work across the whole research lifecycle?

Workshop identified:

1. There is no single common e-Infrastructure
2. Ease of use is the initial barrier
3. Dealing with complexity is complex
4. Trust is important
5. Open development is necessary
6. Give credit for digital creation
7. Attitudes must be changed

~70 attendees (developers, PIs, managers, researchers and funders)

http://www.nesc.ac.uk/technical_papers/UKeS-2009-01.pdf

Classification of Open Source Business Models

- Development Model
  - ([vendor|community|mixed]/[open source|hybrid])
- Licensing Strategy
  - (Dual | Open-Core | Open and Closed | Single | Assembled | Closed)
- Revenue Trigger
  - Commercial License, Subscriptions, Service/Support, Embedded Hardware, Embedded Software, Software as a Service (SaaS), Advertising, Custom Development, Other Products and Services
- http://blogs.the451group.com/opensource/2009/03/12/a-classification-of-open-source-business-strategies/

Sustainability Models for Research Software

- Grant Mosaic
- Flagship (e.g. CCPs: DL_POLY)
- Institutional (e.g. Subject repositories, CNX)
- Fully Costed (e.g. HECTOR CSE Support)
- Mixed Enterprise / Consultancy (e.g. SugarCRM)
- Foundation (e.g. Sakai, R)
- T-shirt

UK e-Science Core Programme: Business Models for Sustainability (2007)
http://www.jisc.ac.uk/media/documents/programmes/einfrastructure/day2_breakoutbusinessmodels.pdf

Grant Mosaic model

- You manage to divert some of your researchers / grad students to do software development on each project grant
- e.g almost all research software
- Pros: “easy” to get funded
- Cons: hard to get external contributors,

Flagship model

- Where the software is so important for research that core development is easy for funders to justify
- E.g. CCPs (DL_POLY etc)
- Pros: provides long term sustainability for software as long as it needs it
- Cons: it doesn’t scale

Institutional Model

- Software is so useful and prestigious that an individual institution support and integrate into their general infrastructure
- e.g. Repositories, CNX
- Pros: institutions can provide one of the longest term sustainability routes
- Cons: silos, difficulty in open governance

Fully costed model

- Software development is costed as parts of grants at proposal stage
  - "software support credits"
  - E.g. HECTOR Distributed CSE Support
- Requires a change in the funding and a change in attitudes within research groups (I could do that myself)

Consultancy model

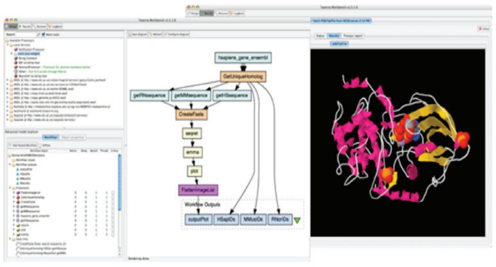
- Where you have good set of basic features but people want more customisation
- E.g. Most CMS software
- Pros: Diverse income streams
- Cons: only works for certain sorts of software, with a large enough community

Foundation model

- There is a central point of coordination and governance. People pay money to have influence.
- E.g. Sakai
- Pros: very good at harnessing community contributions
- Cons: requires effort to set up, needs to be at the right level

Taverna Workbench

- Initially funded through e-Science myGrid project (2001-2005)
- Directly funded through OMII-UK (2006-2010)
    - Plus marketing, outreach, legal and networking
- Platform funding (2009-2014)
- caBIG subcontract
- Eli Lilly development
- 40,000+ downloads of Taverna 1.x
- Take up in other domains, e.g. astronomy

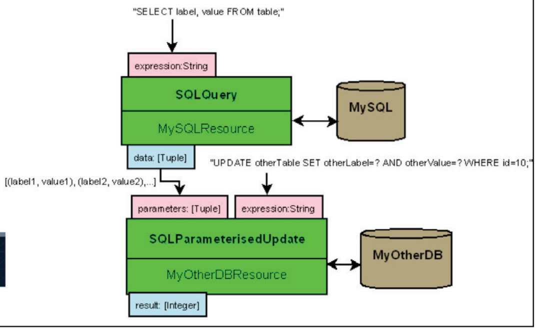

Taverna user Interface with a workflow diagram and the result panel displaying a protein structure.

omii-uk

Web: www.omii.ac.uk Email: info@omii.ac.uk

OGSA-DAI

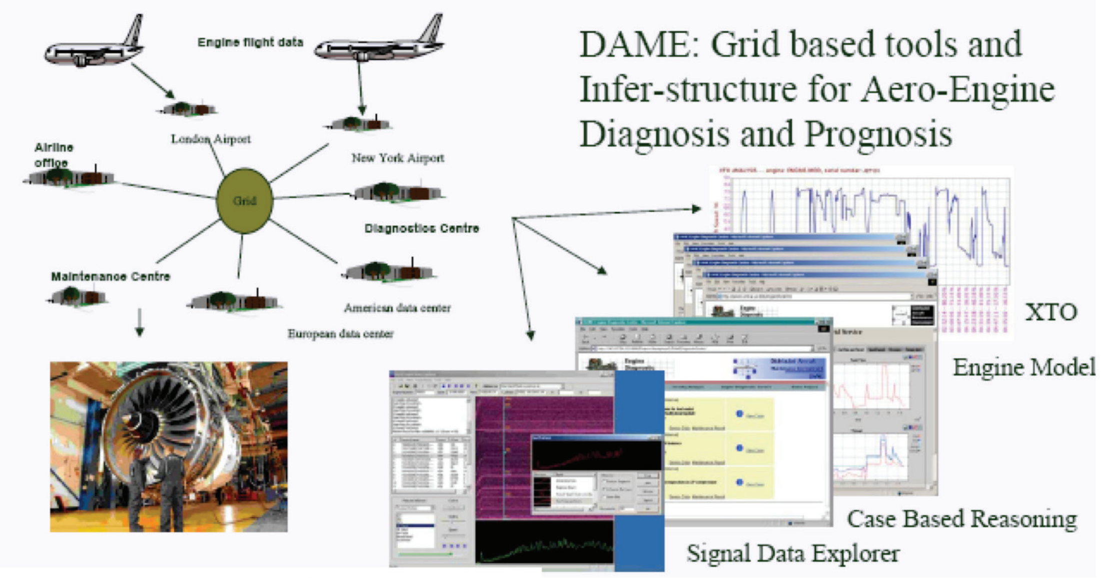
- Initially funded through e-Science Core Programme (2002-2005)
- Directly funded through OMII-UK (2005-2010)
- Additional funding for EC deployments
    - NextGRID, OMII-Europe
- Contributions from
    - Austria, Brazil, Germany, Japan, Russia, UK



omii-uk

Web: www.omii.ac.uk

DAME: Grid based tools and Infer-structure for Aero-Engine Diagnosis and Prognosis



broaden

CARMEN
CODE ANALYSIS, REPOSITORY & MODELLING FOR E-NEUROSCIENCE

Slide from Jim Austin

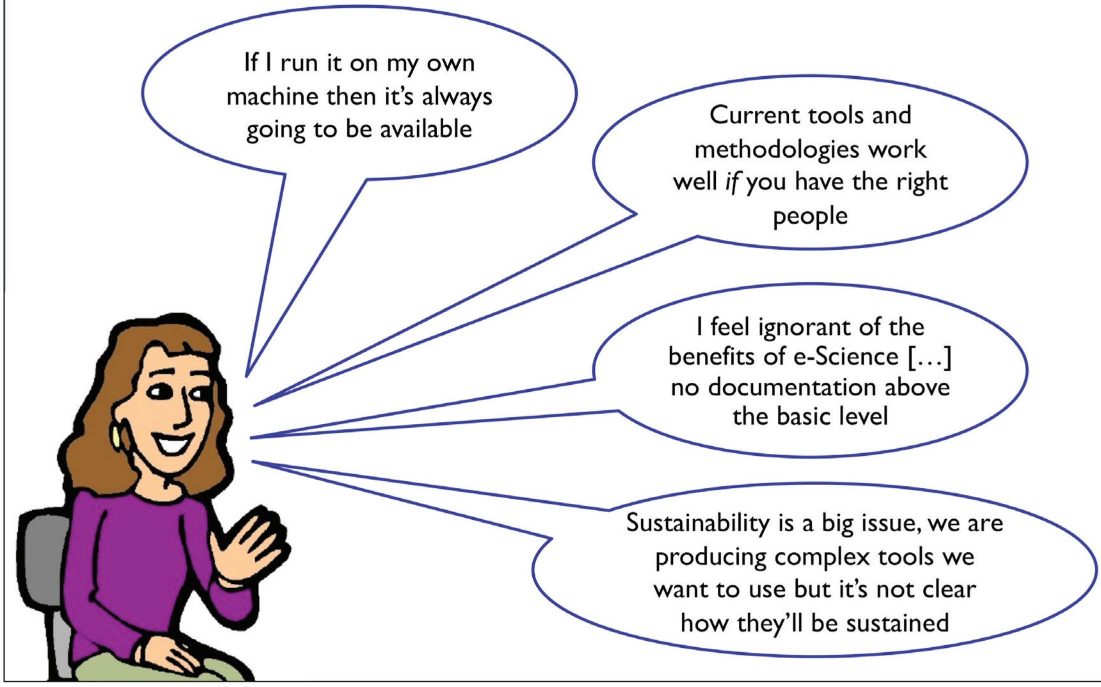
The $5m question: staff vs software

- Can we prioritise software sustainability over staff retention?
  - skilled staff availability to work on projects
- Why get a grant for $3m that builds on someone else's software when you can get a grant for $5m to build your own?

Increasing participation is the key to long term sustainability

So what do people say?

engage
Engaging research with e-Infrastructure


If I run it on my own machine then it's always going to be available
Current tools and methodologies work well *if* you have the right people
I feel ignorant of the benefits of e-Science […] no documentation above the basic level
Sustainability is a big issue, we are producing complex tools we want to use but it's not clear how they'll be sustained

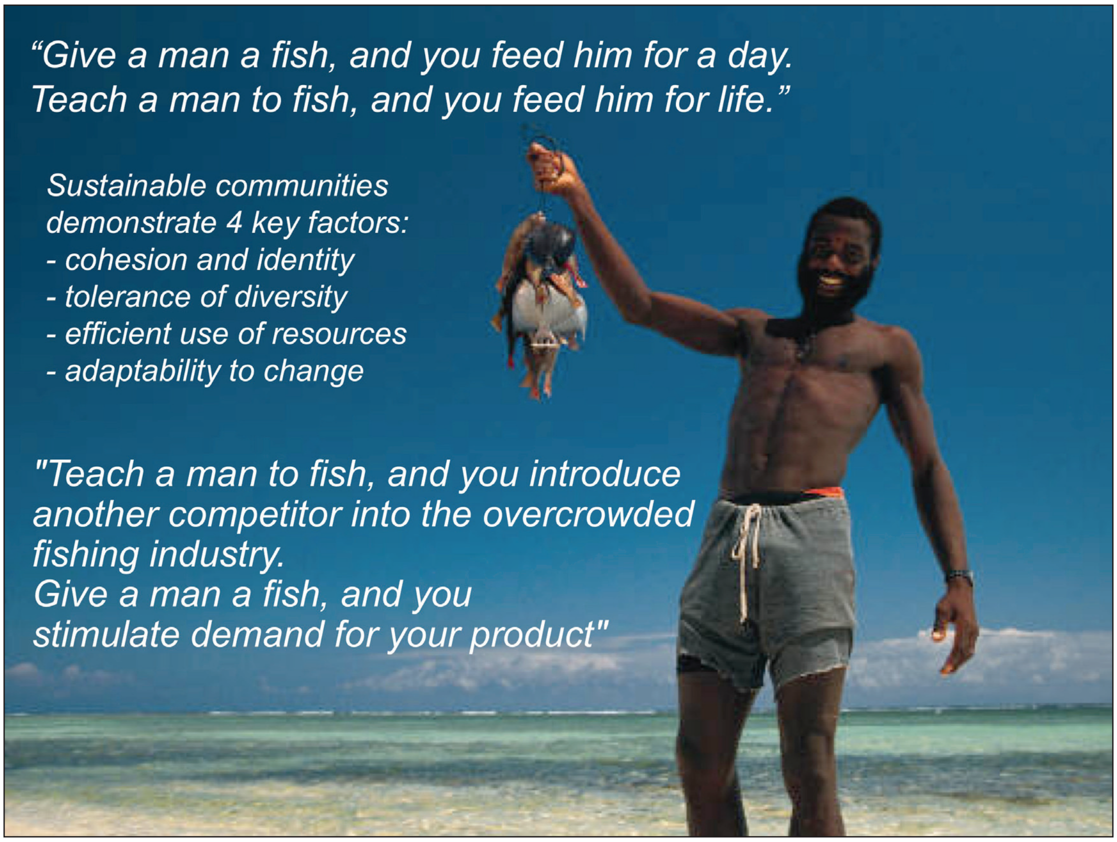
ENGAGE preliminary, non-empirical qualitative finding

People will tend to prioritise *ease of use*, *support* and *continued development* over a complete feature set

*This requires a **sustainable** community around the software and **trust** by the users in the e-Infrastructure providers (and vice-versa)*

The Four Levels of e-Science Enlightenment

- 1) **Resources:** Providing access to a larger and wider diversity
- 2) **Automation:** Repeatability and management of experiments
- 3) **Collaboration:** Intra + cross disciplinary networks
- 4) **Participation:** Increasing access to a wider set of users; increasing knowledge in a domain

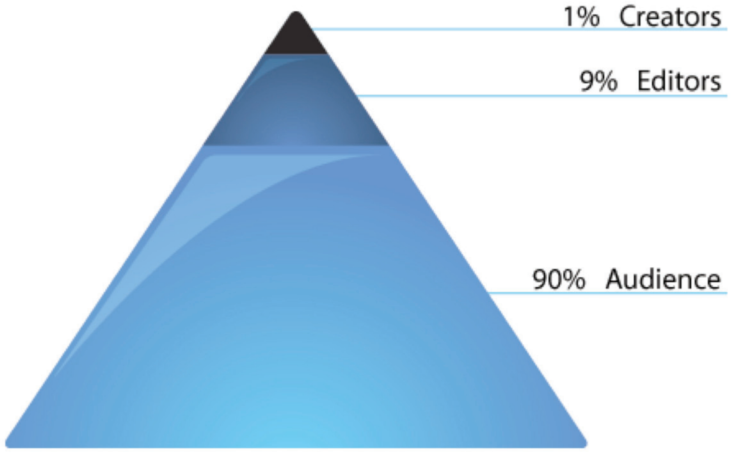


“Give a man a fish, and you feed him for a day.
Teach a man to fish, and you feed him for life.”

Sustainable communities demonstrate 4 key factors:
- cohesion and identity
- tolerance of diversity
- efficient use of resources
- adaptability to change

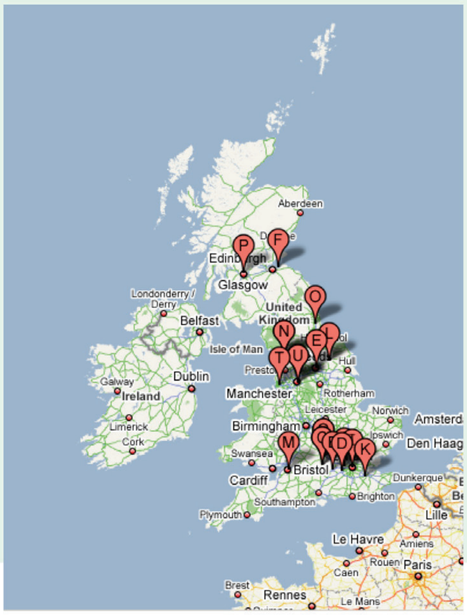
"Teach a man to fish, and you introduce another competitor into the overcrowded fishing industry.
Give a man a fish, and you stimulate demand for your product"

Participation Inequality aka “90-9-1”



(cc) Jake McKee & 90-9-1.com

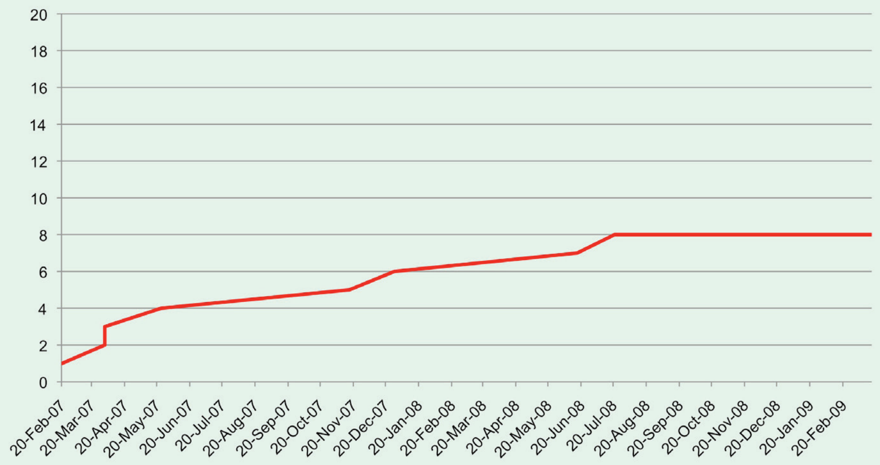
National Grid Service



- Funded by EPSRC and the JISC
- Coherent electronic access for UK researchers to all computational and data based resources and facilities required to carry out their research, independent of resource or researcher location

Growth

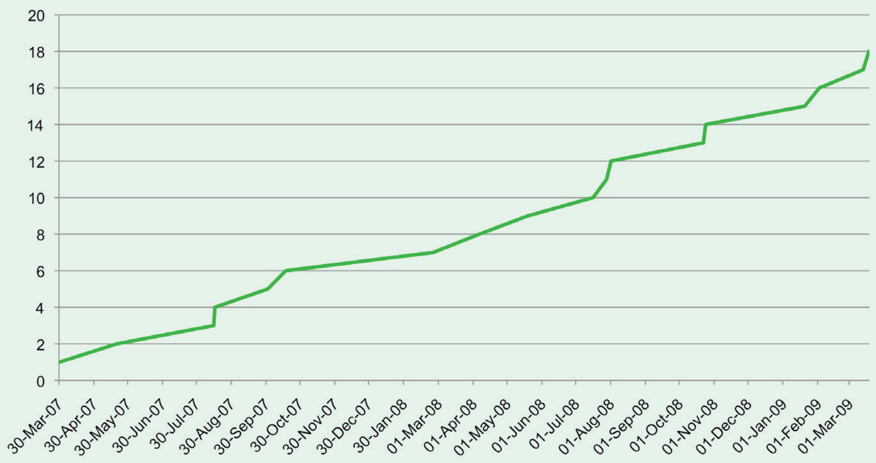
- Partners



NGS National Grid Service

Growth

- Affiliates

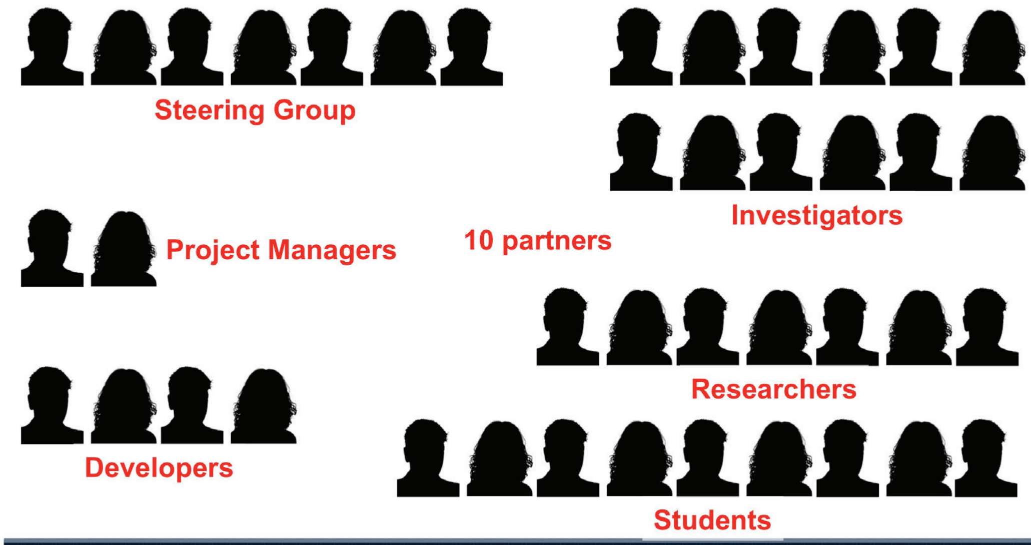


NGS National Grid Service

Lessons Learnt

- Have a clear description of benefits
- Aim for minimal overhead for member sites (affiliate rather than partner as first step, with clear process for upgrade)
- Provide good central support for site contacts
- Supporting Campus Champions will be a good way of spending the underspend…

A typical e-Science project organisation?



Smart Growth through Collaboration

- How do you turn users into collaborators into contributors?
- Moving from a single team at a single organisation to more diverse, sustainable development
- Improve availability and visibility
  - celebrate success in the community

OMII-UK NEWS

www.omii.ac.uk June 2008

Access Grid learns to dance

Dance is fundamentally concerned with the body moving in space and time. New technologies that challenge the traditional definitions of these categories provide fruitful territory, both creatively and critically, for the dance artist-scholar. One such technology is Access Grid, which has traditionally been used for meetings between distant work colleagues, and is now being used by the *e-Dance* project for *telepresent performances*. Interdisciplinary projects, such as e-Dance, exemplify the appeal and benefits of e-Research across all research disciplines.

e-Dance repurposes the Access Grid and integrated Grid technologies, as a context for telepresent performance and hypermedia documentation. In making use of recent developments in the visualisation of spatio-temporal structures and

Continued on page 3.

Catherine Bennett performs an Access Grid improvisation during an *e-Dance* intensive

PAG: Making Access Grid easy

The growth of the Access Grid videoconferencing system has been relentless. Worldwide, the AGToolkit has been downloaded tens of thousands of times. There are now over 120 large, Access-Grid-enabled conference and meeting rooms. However, the uptake of the system could be increased even further by overcoming Access Grid's lack of easy accessibility. Making Access Grid easy is the aim of the PAG (Portlet Access Grid) project, which is funded by OMII-UK.

PAG is a portlet version of the Access Grid Toolkit (AGTk) client, which allows any web-based machine – even one located in a remote place or behind a restrictive firewall – to become an interface to the Access Grid. PAG overcomes the complicated Access Grid installation process by downloading and installing all of the client-side tools automatically using Java Web Start. The PAG project has also been working to remove the restrictions imposed by network firewalls and bandwidth limitations through the implementation of a

Continued on page 3.

T-Rex is coming - page 4

Summer of code - page 2

NGS Roadshow success - page 5

Fighting malaria - page 7

© OMII-UK 2008

OMII-UK Services

- Understand demand
- Increase potential users
- Increase research outputs
- Increase community participation
- Community outreach
- Improve quality of experience + code
- Identify projects and domains
- Governance, transparent development, communication

Creating a Community

- Who are the users of the software?
- Why do they use it?
- What do they value from it?
- What is their relationship between developers and users?
- What do people *want* to do?
  - not how can they use what we've got to do it

Helping establish communities

- Communities require more than functional software
  - documentation and training
  - guaranteed long-term support
  - stable APIs as well as interoperable standards
  - sharing of best practice and issues
- Clear understanding of the requirements that make them a distinct community

Leveraging Infrastructure

- Prevent defects rather than just fix
- Use technology to lower effort
- Keep code shippable to aid collaboration
- Improve design continually and cost-effectively
- Make it easy for new developers
- Consistent pace that balances short- versus long-term requirements
  - a frequent release cycle keeps people engaged

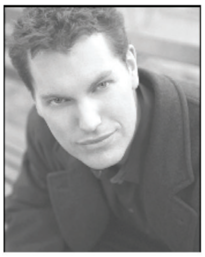
Governing Sustainably

- Copyright
- Licensing
- Value
- Decisions
- Trust

OSS Watch
open source software advisory service

- Create a governance model that makes it easy to contribute, easy to make decisions and maintains quality, whilst being able to adapt to change
- Be prepared to make the gradual transition from benevolent dictatorship to democratic meritocracy

The trade-offs of a larger community



"Connected, distributed systems, from power grids to business firms to even entire economies, are both more fragile and more robust than populations of isolated entities."

Duncan J. Watts
Professor of Sociology
Columbia University

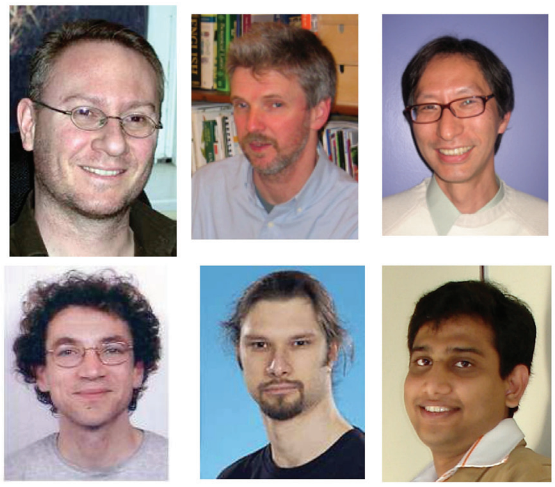
• Sometimes the thing that kills software is that the community becomes too fragile

Community Engagement and Communication

- Training, tutorials, advice
- Beta-Testers and Tech Previews
- User Forums, User Advocates
- Mailing Lists, IRC, online forums

- Different ways of understanding community needs and gathering feedback
- Multiple approaches from multiple skill bases to attack a common problem
- Make people feel a part of the team, and the team will grow

PALs (Product/Area Liaisons)

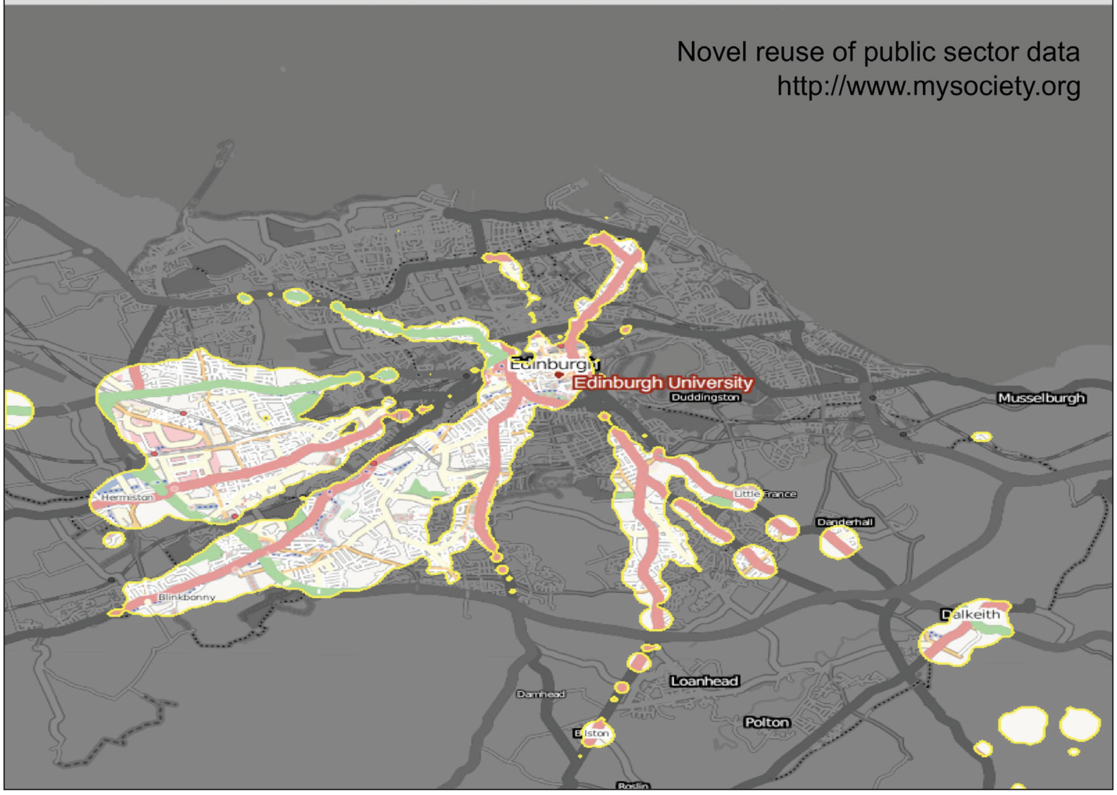


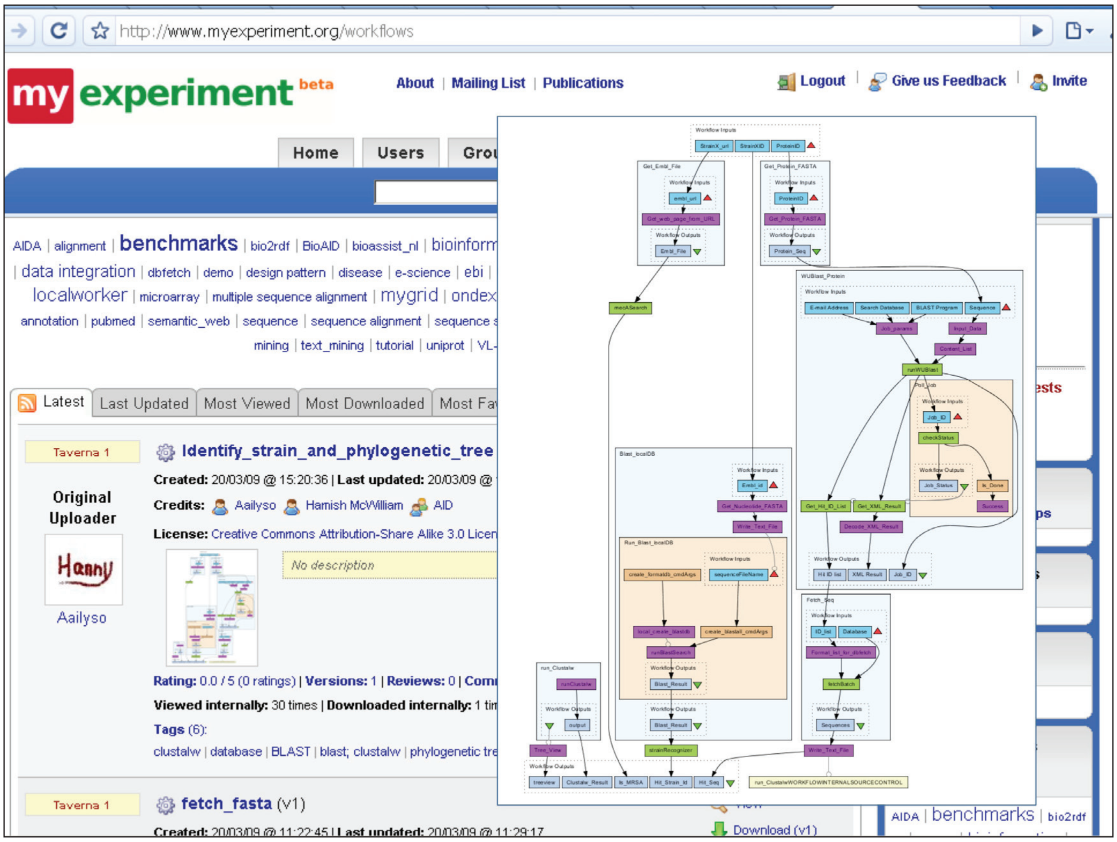
- Eyes and ears in the community
- Funding for travel
- Priority access
- Reporting in return

Can we still consider traditional notions of software?

The light, coloured areas of the map represent places where it's faster to use public transport than to drive if you want to get to work in central Edinburgh by 9AM (centred on postcode: EH1 2QL)

Novel reuse of public sector data
http://www.mysociety.org





http://www.myexperiment.org/workflows

myexperiment beta

About | Mailing List | Publications

Logout | Give us Feedback | Invite

Home Users Grou

AIDA | alignment | benchmarks | bio2rdf | BioAID | bioassist_nl | bioinform
| data integration | dbfetch | demo | design pattern | disease | e-science | ebi |
localworker | microarray | multiple sequence alignment | mygrid | ondex
annotation | pubmed | semantic_web | sequence | sequence alignment | sequence s
mining | text_mining | tutorial | uniprot | VL-

Latest | Last Updated | Most Viewed | Most Downloaded | Most Fa

Taverna 1

Original Uploader

Aailyso

Identify_strain_and_phylogenetic_tree

Created: 20/03/09 @ 15:20:36 | **Last updated:** 20/03/09 @

Credits: Aailyso Hamish McWilliam AID

License: Creative Commons Attribution-Share Alike 3.0 Licen

No description

Rating: 0.0 / 5 (0 ratings) | **Versions:** 1 | **Reviews:** 0 | **Com**

Viewed internally: 30 times | **Downloaded internally:** 1 tim

Tags (6):

clustalw | database | BLAST | blast; clustalw | phylogenetic tre

Taverna 1

fetch_fasta (v1)

Created: 20/03/09 @ 11:22:45 | **Last updated:** 20/03/09 @ 11:29:17

Download (v1)

AIDA | benchmarks | bio2rdf

Facebook application
2.5 million active users
Pay for "respect"
$1m / month turnover

http://www.developeranalytics.com/2008_08_facebook_apps_making_more_money.php

UK Industrial IT Projects

- "Today, only 30 percent of government IT projects and programs are successful. We want 90 percent by 2010/11"
  - Joe Harley, CIO UK Dept. Work and Pensions
- £12.7bn **National Programme for IT (NPfIT)**, three suppliers have walked away

Free is a hard price to beat

- What happens if universities just give away their research?
- Save £160m a year just on journals.
- Imagine what we could save if we actually band together
- http://www.jisc.ac.uk/publications/publications/economicpublishingmodelsfinalreport.aspx

Software as a shared facility

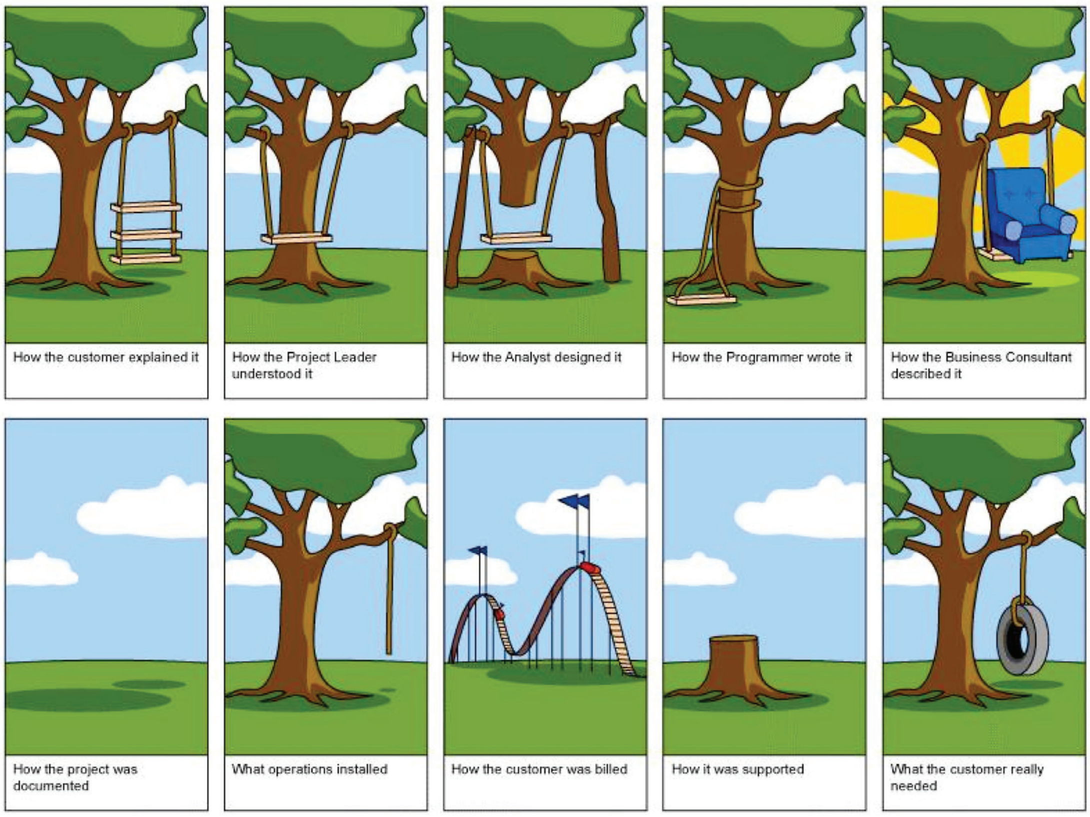
- Community involvement is the route to long-term sustainability
  - but grow a community too large and it lacks cohesion
- OMII-UK tries to identify and generate synergies amongst different groups
  - identify specialisms which are useful across disciplines
  - create benefit without diluting community vision
  - provide networking and sharing of best practice
  - help with the unglamourous/specialist work
- Centralise software sustainability as a facility?

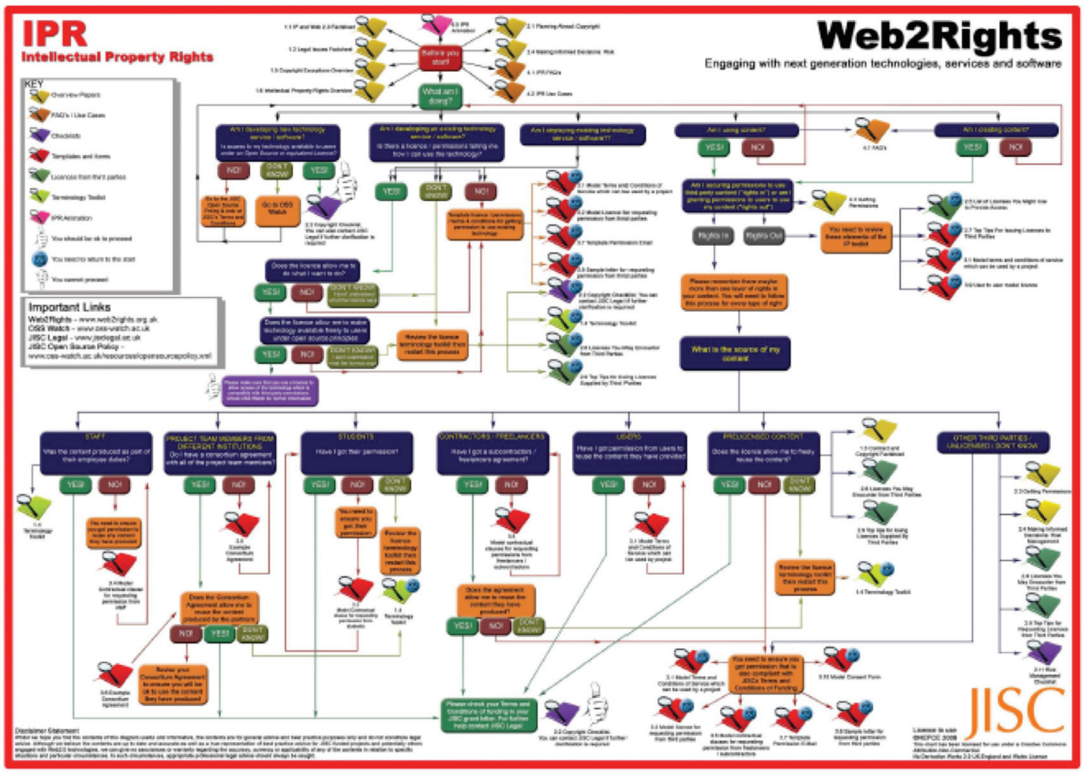
Thank You!

omii-uk Software Solutions for e-Research

EPSRC Software Sustainability

- Provide a research infrastructure which will aid the long term sustainability of software which enables high quality research.
  - Development of software to an acceptable quality for wider deployment, through the application of additional software engineering to prototype software delivered by UK research projects.
  - The maintenance of software which enables high quality research, through the management of a repository for selected software.
  - Community outreach and promotion to ensure effective uptake of the services that the infrastructure will provide.
  - Engagement with the international community, for e.g. through the dissemination of e-research software, establishing best practice and standards, providing internationally recognized codes


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed


IPR
Intellectual Property Rights
Web2Rights
Engaging with next generation technologies, services and software
KEY
Overview Papers
FAQ's / Use Cases
Checklists
Templates and forms
Licences from third parties
Terminology Toolkit
IPR Animation
You should be ok to proceed
You need to return to the start
You cannot proceed
Important Links
Web2Rights - www.web2rights.org.uk
OSS Watch - www.oss-watch.ac.uk
JISC Legal - www.jisclegal.ac.uk
JISC Open Source Policy - www.oss-watch.ac.uk/resources/opensourcepolicy.xml
Before you start
What am I doing?
Am I developing new technology service / software?
Am I developing an existing technology service / software?
Am I deploying existing technology service / software??
Am I using content?
Am I creating content?
YES!
NO!
DON'T KNOW
Rights In
Rights Out
What is the source of my content
STAFF
PROJECT TEAM MEMBERS FROM DIFFERENT INSTITUTIONS
STUDENTS
CONTRACTORS / FREELANCERS
USERS
PRELICENSED CONTENT
OTHER THIRD PARTIES / UNLICENSED / DON'T KNOW
JISC

Web: www.omii.ac.uk Email: info@omii.ac.uk