Report of the 3rd DOE Workshop on HPC Best Practices: Software Lifecycles

San Francisco, California

September 28-29, 2009

HPC Best Practices: Software Lifecycles

Findings and notes from the the third in a series of HPC community workshops

Workshop Sponsors: DOE Office of Science – ASCR and DOE NNSA

Chair: David E. Skinner (LBNL)

Vice-Chair: Jon Stearley (SNL)

Report Editors: John Hules (LBNL) and Jon Bashor (LBNL

Attendees/Report Contributors:

Deb Agarwal (LBNL), Dong Ahn (LLNL), William Allcock (ALCF), Kenneth Alvin (SNL), Chris Atwood (HPCMPO/CREATE), Charles Bacon (ANL), Bryan Biegel (NASA NAS), Buddy Bland (ORNL), Brett Bode (NCSA), Robert Bohn (NCO/NITRD), Jim Brandt (SNL), Jeff Broughton (NERSC), Shane Canon (NERSC), Shreyas Cholia (NERSC), Bob Ciotti (NASA NAS), Susan Coghlan (ANL), Bradley Comes (DoD HPCMP), Paul "Steve" Cotter (ESnet), Nathan DeBardeleben (LANL), Sudip Dosanjh (SNL), Tony Drummond (LBNL), Mark Gary (LLNL), Ann Gentile (SNL), Gary Grider (LANL), Pam Hamilton (LLNL), Andrew Hanushevsky (SLAC), Daniel Hitchcock (DOE ASCR), Paul Iwanchuk (LANL), Gary Jung (LBNL), Suzanne Kelly (SNL), Patricia Kovatch (NICS/UTK), William Kramer (NCSA), Manojkumar Krishnan (PNNL), Sander Lee(NNSA), Janet Lebens (Cray Inc.), Lie-Quan Lee (SLAC), Ewing Lusk (ANL), Osni Marques (DOE ASCR), David Montoya (LANL), Krishna Muriki (LBNL), Tinu Ogunde (ASC/NNSA/DOE), Viraj Paropkari (NERSC), Rob Pennington (NSF), Terri Quinn (LLNL), Randal Rheinheimer (LANL), Alain Roy (Univ. Wisconsin-Madison), Stephen Scott (ORNL), Yukiko Sekine (DOE ASCR), David Skinner (LBNL), Rebecca Springmeyer (LLNL), Jon Stearley (SNL), Krystyne Supplee (Cray Inc.), Henry Tufo (NCAR), Craig Tull (LBNL), Andrew Uselton (NERSC), Daniela Ushizima (LBNL), Tammy Welcome (LBNL), Vicky White (ORNL), Dean Williams (LLNL), Kathy Yelick (NERSC), Mary Zosel (LLNL)

Introduction

The third in a series of best-practices workshops, this workshop brought together experts in HPC libraries, tools and systems to identify best practices for creating and maintaining reliable and sustainable software for use at HPC centers as they enter the petascale era. Not only did the workshop look at software within those specific infrastructure layers, but also the interactions between the layers. Also present were managers who must prioritize the most effective ways to support the most useful software for their users and center operation in the face of budget constraints. Rather than focusing on software development or software maintenance, the scope of the workshop included processes, standards and policies throughout software lifecycles – from origins that motivate its development to long-term pervasive use in production computing settings. The workshop was attended by 70 representatives from 18 HPC centers and offices.

About the Workshop Series

This workshop was sponsored by the SC/Advanced Scientific Computing Research (ASCR) program and the National Nuclear Security Administration (NNSA)/Advanced Simulation and Computing (ASC) program. This workshop series assess current and emerging techniques, practices, and lessons learned for effectively managing leading-edge computing systems at high-performance computing centers (HPCCs). This third workshop in the series was hosted by NERSC with assistance from Sandia National Lab.

Organization

This workshop was organized as a community effort and conducted with the inclusion of a broad range of inter-agency HPC centers and stakeholders. This approach was important both in bringing the community together to discuss comon issues and to recognize other, similar efforts to address the needs in the HPC software space. These include meetings on software resilience (http://institutes.lanl.gov/resilience/conferences/2009/), technology auditing (XREF), and software sustainability (http://cisoftwaresustainability.iu-pti.org/).

The scope of this workshop was purposefully limited to software deployed *by* HPC centers for general purpose use. The space of application software needs and scientific computing applications *as software* was deemed sufficiently complex that it could not be treated in just a two-day workshop. Thus, the workshop organizers focused on the software components in wide use among science teams using HPC resources, the software that HPC center staff have a stake in, and HPC systems-facing sofware.

This two-day workshop devoted the first day to software by categories, while the second was spent looking at the various stages of software development. Using breakout groups specific to each area allowed for treatment of issues specific to certain types of software,

as well as cross-cutting issues in the HPC software ecosystem. In addition to the breakout sessions, the workshop included a set of plenary discussions. Here is a list of the breakout sessions:

Software Categories (day 1)	Software Stages (day 2)
tools libraries system management system sofware	planning development integration sustainment

Each breakout session was asked to consider the following cross-cutting questions:

- What are the best practices and tools?
  - Inside HPC
  - Outside HPC
  - Education and Training
- What are the top challenges?
- What new technologies are needed?

Note: Software specific to individual science domains and scientific applications themselves are certainly worthy of similar treatment, but devising best practices for that space would require expert advice from each of the science domains in question. This is worthy of pursuit in a follow-on workshop, perhaps deriving a jump-start from the various Greenbook (http://www.nersc.gov/news/greenbook/) actvities currently underway in the DOE Office of Science.

Findings and Documentation

The workshop resulted in the following three findings. More information on each finding is presented in the subsequent sections.

1. Software should be recognized as **an enduring facilities concern**, requiring ongoing resource investment at all stages of the software lifecycle.
2. The community would be well served by an **online catalog of HPC software** for sharing information and needs in the software space.
3. **Allocation of compute time and access to test systems** for software development and maintanence is currently under-served.

In addition to this report, the following documents were produced as a result of this workshop:

- working notes and presentations at the workshop website: http://outreach.scidac.gov/swbp
- a survey providing feedback about the workshop
- a revised inventory of prioritized software needs

Workshop Overview

The HPC science community is increasingly benefitting from and dependent upon shared software stacks. Examples include libraries enabling advancements in multiple scientific domains, workflow tool consolidation among sites facilitating portability, and the increasing ubiquity of Linux. While this trend presents valuable scientific and economic advantages, it is increasingly important that we examine best practices for building solid and reusable software foundations for HPC centers.

The workshop was attended by 70 representatives from 18 HPC centers and offices. Although the workshop was centered primarily around specialized breakout sessions, each day's program began with plenary sessions to help frame discussions among the smaller groups.

On the first day, introductory keynote addresses by Rusty Lusk of Argonne National Lab and Dean Williams of Lawrence Livermore National Lab struck complementary perspectives on how software develops. Lusk, who is a leading member of the team responsible for MPICH implementation of the MPI message-passing interface standard, noted that "Hardware is soft and software is hard. It creates something out of nothing." Lusk focused on the MPICH project, describing how it has evolved over the years from a research prototype to being a ubiquitous software standard widely used in production HPC. MPICH is a software library and cluster of accompanying tools that grew from a research effort targeting parallel computing through message passing. Now in wide use and supported by HPC centers and vendors alike, one attribute of the MPICH project that has led to its implementation and sustainment is that from the beginning, the researchers/developers targeted production computing settings as the aim of the project. Careful attention to balancing research and production computing needs is one of the factors leading to MPICH2 winning a R&D100 award. Lusk summarized the best practices which led to production computing usage as being robustness, portability, performance, and ease of extension and customization.

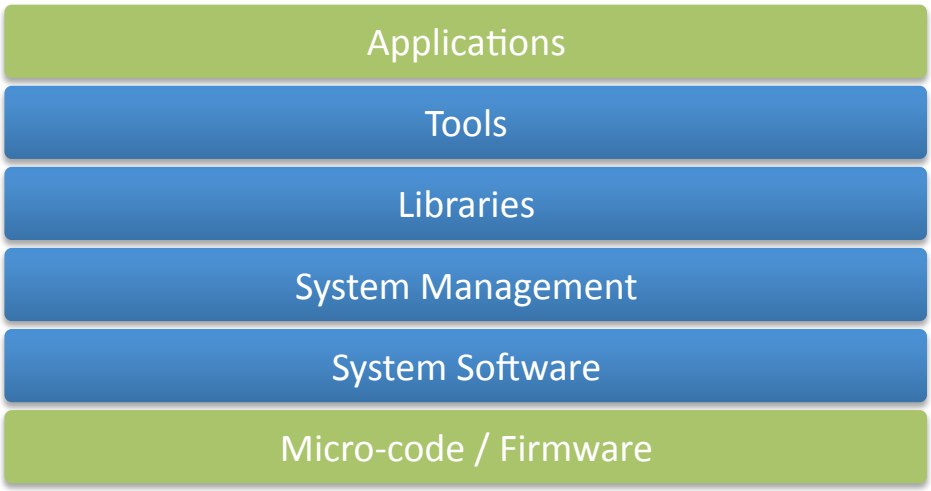
Dean Williams, principal investigator of the SciDAC Earth Systems Grid (ESG) project, described ESG, which aims to provide infrastructure to enable faster, easier sharing of climate change research data for the world's climate research community. ESG provides data-as-a-service for a wide variety of climate, geophysical, and biological data to its 2,300 registered users. Although it has achieved success in a number of areas, ESG faces significant challenges in coming years as the size, complexity, and number of climate datasets grow dramatically. To address these hurdles, the ESG teams aims to broaden the project to support multiple types of model and observational data, provide more powerful (client-side) ESG access and analysis services, enhance interoperability between common climate analysis tools and ESG, and enable end-to-end simulation and analysis workflow.

The second day of the conference opened with a plenary panel discussion on software best practices issues. Osni Marques detailed the ongoing DOE SHAIP effort to sustain and maintain reliable software tools for scientists. William Kramer gave a report from the ongoing Internation Exascale Software Project (IESP), which released a draft report at

SC09 (http://www.exascale.org). IESP ambitiously intends to organize international agencies around sharing costs and leveraging each other's work in the short, medium, and long term. Tony Drummond gave a review of the DOE ACTS Collection, which curates and educates in the area of HPC software.

The Function (and Malfunction) of Software in a Production Computing Environment

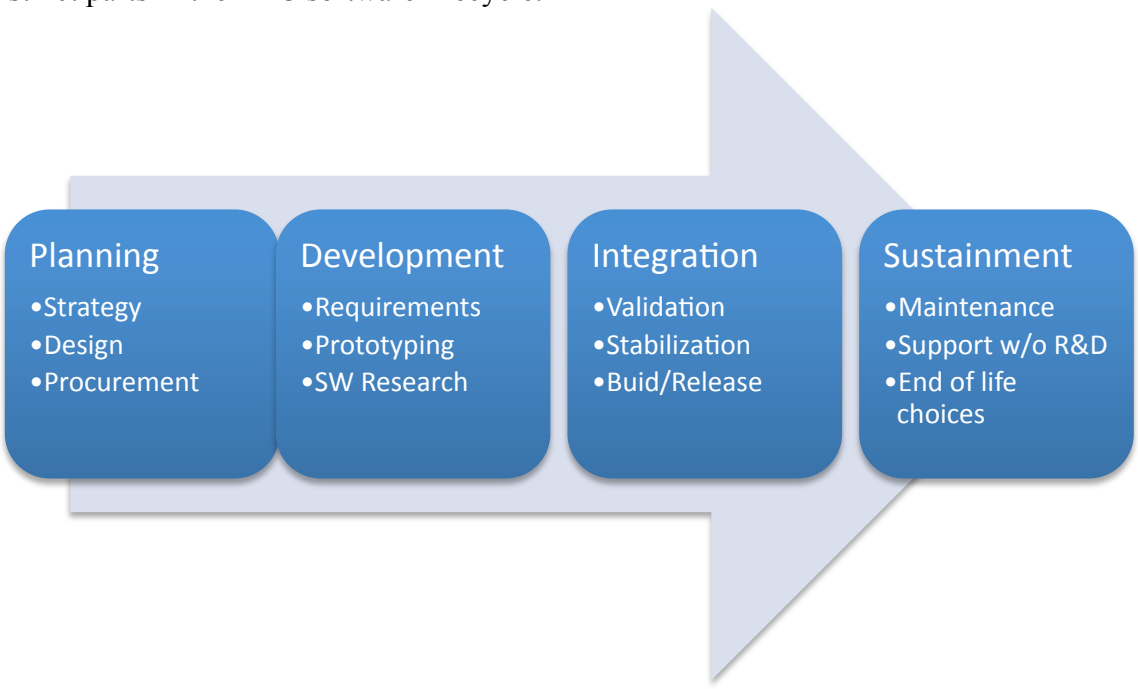
In order to set the scope of this workshop, which was mainly attended and sponsored by those involved in procuring and managing high performance computing centers, we first had to define the type of software we are addressing. This workshop divided the HPC software ecosystem into the following strata:



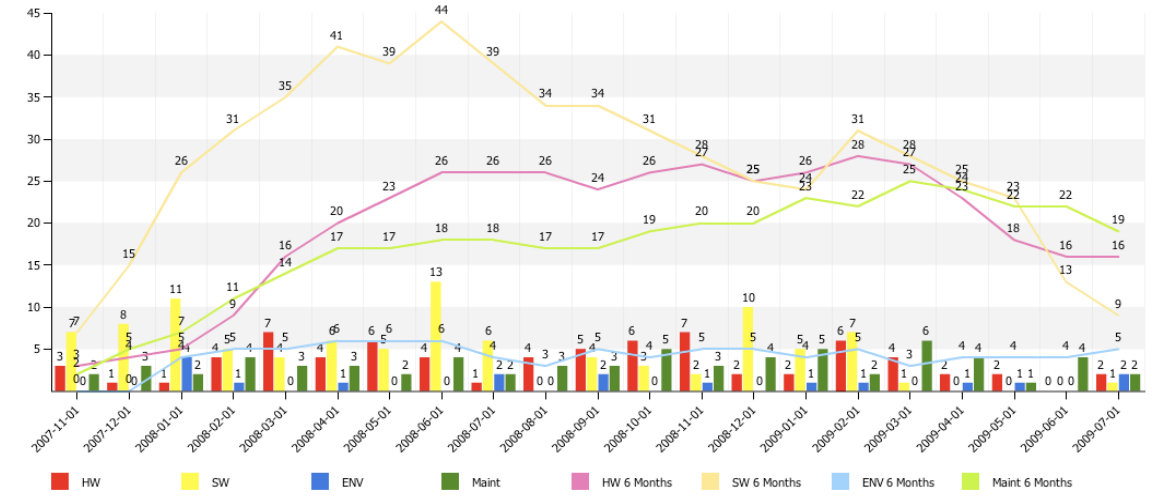
and identified the middle four layers as the scope of what HPC managers might best address as software best practices. The top and bottom software functions are, at many HPC centers, beyond the scope of managers or staff to control. Applications often come already developed from our customers. Very low-level software is often outside the control of those who purchase, rather than develop, HPC hardware. These circumstances vary between centers, and our choice to limit the scope of what we mean by *software* is meant to balance the discussion on common ground rather than to exclude topics outside the four areas above. Certainly the topic of HPC best practices in application-level software could comprise in itself an equally large or larger set of discussions.

This software stack's top-level function, through the execution of application code, is to make computing platforms a useful tool for science. The underlying layers, which should function reliably with good performance, all serve that end goal, and the degree to which they function depends largely on integration and maintence of software. An a priori postion of the attendees, based on decades of experience within HPC, is that both hardware and software systems often do not function out-of-the-box (http://www.nersc.gov/projects/HPC-Integration/) but instead require significant attention at all stages of the software lifecycle.

Software lifecyles vary from faciltiy to facility. This workshop categorized four phases as distinct parts in the HPC software lifecycle.



The term *production* is often used in a rather loose way to connote a stable computing platform which experiences infrequent failures or interruptions. Managers of HPC centers most often take a metric-based approach to indentifying the frequency, duration, and cause of interruptions in service. Much of the discussion and body of this report focuses on practices which drive down malfunction rates in HPC software.



System outages for a 20K core machine over two years sorted by outage type.

In short, when software does not function, HPC does not function. Increasingly the preponderance of interruptions in HPC systems are caused by software faults rather than

hardware. The time is right to reconsider how we expect software to be developed, integrated, and maintained. A model that expects software, once developed, to function evermore for production needs is increasingly unrealistic in the face of the rising complexity of HPC systems. One implication of this is that if HPC software is to be a research driven endaevor, i.e., that the software is the fruit of computer scicence and applied math research, then the later stages of software's use in a production HPC setting may be critically under-resourced.

Workshop Priority Summary Findings and Recommendations

As might be expected, workshop participants developed many suggestions in the course of their discussions. Participants then evaluated the priorities for all the recommendations and agreed on these three highest priority areas that will determine the success of petascale systems:

1. Recognize that software is an enduring facilities concern.

Recommendation: As long as software is treated as a research concern, it will only be the rare and exceptional cases that make the transition from the research group doing the development to providing sustained value in the broader HPC ecosystem. In evaluating software from a facilities perspective, the following points should be considered:

- Software serves roles in both research as well as in facilities. As such, it is worth supporting in both contexts and with clear delineation of roles and responsibilities.
- Motivate software research groups to target production use when addressing facilities needs or planning for facility-level implementation.
- Prototype software is important but does not close the loop on delivering useable value to HPC customers.

Note: This recommendation overlaps somewhat with findings of the NSF through their reports at http://cisoftwaresustainability.iu-pti.org/

2. Develop an online catalog of available HPC software which describes its use, current state and stakeholders.

Recommendation: Managing software and making sound judgments about software deployment is enhanced when information about software is shared with the wider community. A community catalog would allow for better choices to be made, better collaboration on software projects, and a lower level of overall effort in supporting software. This catalog should define:

- The role of the software in HPC
- The stakeholders and supporters of the software, including the nature of the support
- A list of ongoing and needed actions for the software

3. Provide allocations of computer time and improved access to test systems to make software perform better once it goes into production.

Recommendation: Performance expectations based on extrapolations from software research and design must be met with observations from their implementations in a production computing center. The best way to resolve the disconnect between software that works in R&D but does not in production HPC is for R&D software testing to move into the production setting. This requires a significant outlay of computing allocation to provide regular and ongoing testing of software at scale.

- Testing can be developer driven (dedicated time slots for developers) or center driven (regular software tests mixed into the workload with results fed back to developers). Both are useful.
- If a regime of production software testing is not implemented, users from science teams become the de facto software testers.
- These allocations require different metrics for success than other HPC allocations: performance and stability gains, not science delivered.

High Level Concerns Matrix: Top two concerns from each break-out

Software Categories	Software Stages
Tools: Performance at scale and ease-of-use by domain scientists	**Planning:** Technology unknowns and science impact unknowns
Libraries: Testing/maintenance and programmability	**Development:** Software testing and layered change management
System Management: Machine complexity and change management	**Integration:** Software testing in production and vendor/center responsibilities
System Software: Exascale architecture challenges and test system access	**Sustainment:** Funding and community collaboration

Additional key observations made by the attendees:

1. The problem of HPC software is too big for any one site to solve alone.
2. Increasingly it is software failures that impede progress in scientific computing.
3. The software lifecycle deserves attention at every step.

Breakout Session Summaries

As described above, the workshop was focused around eight breakout sessions, each focusing on different aspects of software development and support. At the end of the workshop, each group was asked to submit a report on their findings, including best practices identified by each group. As each group was run by volunteer chairs, the group reports vary in length and detail. Yet each one provides important perspectives on addressing the big picture of improving the state of HPC software development and support as the computational science community moves into the era of petascale computing.

Day 1: Software Layers
Chair: Becky Springmeyer, LLNL

Breakout Group 1: Tools

Group leaders: David Skinner, NERSC; Chris Atwood, DOD

Modern HPC architecture trends have aggressively pushed parallelism to new extremes. Million-way parallelism is not far off. How does this trend impact the usability, or even applicability, of HPC tools? What are tools used for and how do we derive the most valuable methods or use-cases for which tools enable computing at new scales? What is the taxonomy of tools in use and where are there gaps or redundancies?

Cross-Cutting Questions

- What are the best practices and tools? Inside and Outside HPC?
- What are the top challenges?
- What new technologies are needed?

Scope and Taxonomy

What are HPC tools? What are their goals? Tools are programs and libraries used to *develop, deploy, run and understand* HPC application codes, workflows and datasets.

We expand on this taxonomy below, including some information gathered by consensus from the breakout group as to both the importance and the urgency of each class of tools. Importance means that a tool is widely used and relied upon. Urgency indicates that there is an existing or imminent need to develop or adapt a tool for the stated purpose.

Value	Importance	Urgency
1	Vital	Urgent
2	Needed	Some Urgency
3	Desired	Adequate

For each class in the tools taxonomy, we list examples followed by ranked needs. The first number is the importance, the second is the urgency. Finally, examples of known software are given in brackets at the end of each line.

- Develop
  - Compilers: Turn source code into executables (1,2-3, support for hybrid architectures) [PGI, Intel, Cray, gcc, IBM,UPC]
  - Configure/build tools and code generators (2,2, cross compilation) [autotools,make,cmake]
  - Static analysis and document generation tools (3,3, OSS solutions exist) [doxygen]
  - Debugging : Find/reveal errors in programs (1,1, scalability, code and programming model complexity, ease of use) [totalview, DDT, gdb, hpctoolkit, dbx, stat]
- Deploy
  - Issue/bug tracking (1,3, OSS solutions exist) [trac ,jira, roundup, gforge]
  - Code coverage/unit/build test tools (2,2,social as well as technical challenge, ease of use)
  - Performance engineering/debug (1, 1, scalability, overhead, ease of use, programming model complexity) [TAU, OpenSpeedshop, hpctoolkit, gprof]
- Run
  - Workflow and task management (3, 1, move away from duct tape solutions, a lot of room for improvement)
  - WAN/storage data movement (1, 2, data volumes exploding, users want filesystem-like experience)
- Understand
  - Visualization and analytics (1, 2, existing tools need better scalability)
  - Application performance profiling (HPM, MPI, I/O) (1,2, need greater deployment, ease of use, low overhead at scale) [NWperf ,IPM]
  - Progress monitoring (2,2,research needed to recognize problems) [mojo, iowatchdog]

Proposed Best Practice 1: Develop robust, portable, focused libraries and APIs to underpin tool development. This is the lesson of PAPI, avoid having each team of tool developers implement the bridge between low-level architecture details and high-level user questions. Advocate that vendors provide well-thought-out, user-focused, reliable streams of collecting information from their hardware.

Proposed Best Practice 2: HPC software at every level should make clear what information and services it provides to software at higher, lower, and adjacent levels. Discovery of sources of profiling data, resource utilization, and status through systematic planned mechanisms is strongly preferred to silent sources of unpublished information. This benefits both tool developers and users as well as overall resiliency.

Proposed Finding 1: The most significant impediments to the successful development and productive use of HPC tools come from four factors.

1. *Scalability:* massive concurrency brings massive challenges. Tool overhead may scale intractably on 10K–100K tasks. Early access to systems is a perennial challenge for tool developers to ensure that tools can scale to larger numbers of processors.
2. *Programming model complexity:* hybrid/heterogenous architectures, multiple languages and multiple parallel models present a combinatorial challenge for both tool developers and tool users.
3. *Ease of use:* tools are not inherently interesting to most users. The tool community must get past trying to convince users that tools are interesting. Instead, identifying users' needs and providing solutions to their problems is key. Hard-to-use tools will hardly be used.
4. *Data volumes:* Moving from gigabytes to terabytes crushes the performance of tools designed for megabytes.

Proposed Finding 2: A convergence between application and system monitoring of performance is increasingly obvious. As concurrency increases, the likelihood of delivering a million identically performing cores to an application decreases unless a connection is made between individual node health, system-wide resource usage, and application performance.

Proposed Finding 3: The HPC community should develop an online catalog of software used in the community (using OpenID? and a wiki/CDE). This clearinghouse would take in information about HPC tools, libraries, and system software. Registered users could contribute information about the HPC software ecosystem, tracking details such as versions in use, software taxonomy with dependencies, news/discussions, links to project pages, links to software funding opportunities for both researchers and vendors. A more aggressive approach would connect this HPC catalog to svn/cvs repos and deliver nightly/weekly build test data.

Open questions:

1. How do people use tools? Who uses HPC tools? How do we track tool use?

   HPC center staff or users (sometimes vendors), but the level of usage is rarely tracked in a way that provides concrete answers as to the demographics of tool use. Two possible methods for improving the tracking of tools use are as follows:
   - Wrap the modules environment to count invocations of a tool
   - Systematic process accounting reviews, integrated with batch data

2. Is user education about tools really an issue? Marketing models for HPC tools? How to market HPC tools outside HPC?

   User Education/Marketing models: Tools are overlooked sometimes from users because of the following factors:
   - Not aware of functionality that would be beneficial to them

- Tools are not easy to use — a learning curve that individuals are not willing to invest in
- Tools are not implemented in ways that could be beneficial to users.

To move this area forward would require tool developers/supports to be more involved with their user/potential user communities and modify tool capabilities to match and also to have a solid support/documentation environment to make deployment easier.

3. How do tools get developed?
   - How much by planning?
   - How much by necessity?

4. Are there DOE practices (IP constraints, security, etc.) that impede tool development? How do they vary between HPC centers?

5. Which tools can vendors not supply?

   Software solutions that connect directly to the local environment are often better implemented at a local level. Concrete examples of such tools are the interfaces that users and their batch jobs may use to inquire as to the remaining balance of hours in an allocation. Tools that check system status prior to taking compute or data actions are likewise tightly coupled to the local center structure.

   Performance profiling tools that are tightly connected to a hardware vendor can provide in-depth performance information. However, there is also a need to provide high-level performance information across many architectures and across many years. Decoupling performance profiling tools from architectural and vendor specifics makes inter-archtectural performance comparison more stratightforward. A similar argument holds for workload profiling tools, which must span large spaces of applications and architectures.

6. How do tools get maintained? How do you do build/test tools? How are tool releases managed? Who does releases and how often?

   The funding organization of the tool development team forms a Change Control Board (CCB) with designated chair, and key stakeholders from (1) development, (2) quality control, and (3) targeted user community.

   The CCB is the key mechanism to communicate and enforce the linkage between the requirements holders and the development process. Prior to each release cycle, a concept proposal is presented to the CCB, with traceability back to the requirements of the tool, the threshold capabilities addressed with the proposed release, the quality measures that would be met with the release, the proposed schedule, and the risks with recommendation mitigation plan. The CCB will vote go or no-go at this stage to proceed to development. Prior to release of the software, the CCB will again meet to assess whether the threshold measures of the release have been met, which may

include beta usage feedback, bug status, platforms supported, software dependences, training, and user support plans. The CCB will vote go or no-go at this stage to proceed to release. The frequency of the releases may range from weeks to months, depending on the category of tools and the needs of the targetted user community. Periodically, the funding organization and stakeholders may decide to assess progress towards success measures, for example usage of the tools by targetted programs. Furthermore, these measures may be Specific, Measurable, Achievable, Relevant, Time-Bounded (SMART).

7. How can tools be made more modular?

   This is an effort that would need buy-in from the larger community. A standard approach to interfaces would support classes of tools that could be interchanged. Flexible frameworks that integrate tools and make it easier to take advantage of specific tool capabilities in a more scalable structure would allow tool makers to focus on specific capabilities.

8. Which libraries provide value to the tools community? What do HPC tools depend on?
   - PAPI and PAPI-C (need energy and thermal components)
   - DynInst / Stackwalker API
   - MRNet, http://www.paradyn.org/mrnet/
   - LaunchMon, http://icl.cs.utk.edu/~mucci/monitor/
   - PMPI and other name-shifted interfaces POMP, OPARI
   - Libmonitor, http://icl.cs.utk.edu/~mucci/monitor/
   - HWLoc, http://www.open-mpi.org/projects/hwloc/

9. How do we express software dependencies? What are techniques to decrease entropy in software deployment? RPMs, bulletproof installs, etc.? How to expand the adoption of build/test/release processes? How can we test tools in vivo at HPC centers to find breakage before users do?

   Continouous integration solutions are well adopted in the software industry outside HPC. Much could be gained by improving the software engineering practices used within scientific computing. This applies to both software tools as well as user applications.

   Need to develop regression test systems that can be shared among HPC centers. In cases where development is going on, it would be to test the builds — if we need to test across platform configurations, then this needs to be done across organizations. Partnerships needed for this as well as collaboration to build and support the tests. Are there sufficiently general approaches to the above that could be described as best practices?

10. How do HPC centers collaborate (COE, distributed, distributed COES?) in specific tool areas? Identifying organizational core competencies?

Add collaborations to software catalog. There needs to be a need, a core set of individuals that want to make it work, and the environment to collaborate and share. That being said, we have identified during these workshops (tools as well as others). In the case of the tools catalog, this could provide a forum for sharing information, plans, papers, approaches, etc. in regard to a tool or tool category. There still needs to be a driver for these communities to work. A collaboration is only as good as the value that people get from it — if there is no value, it won't be used. Alignment of HPC center tools requirements to leverage and guide vendor efforts?

Breakout Group 2: Libraries

Group leaders: Ken Alvin, SNL; and Tony Drummond, NERSC

Cross-Cutting Questions

- Best practices
  - Automated, integrated build and testing process is key for quality library support
  - Other issues discussed (standards, SQ processes)
- Key challenges
  - Interoperability between existing and emerging programming paradigms (e.g. comm libs, languages, etc.)
  - Sustaining resources for maintenance and support
- New technologies
  - Libraries to insulate applications from new languages and lower-level libraries needed to take advantage of emerging hardware
  - Other code reuse opportunities

Proposed Findings

- The automated, integrated build and testing process is key for quality library support. This is currently done independently by different groups resulting in duplication of efforts. Is the timing right for a cooperative effort?
- There is a need for interoperability between libraries that support existing and emerging programming paradigms.
- There are opportunity for new libraries to shield applications from new languages and lower level libraries that are needed to take advantage of emerging hardware
- Research software that becomes part of the the scientific computing infrastructure requires integrated support and maintenance. This is difficult to reconcile with funding program priorities.
- Testing application versus libraries
  - Scalability
  - Validation
  - Performance
  - Software Dependencies

Breakout Group 3: System Management Software

Group leaders: Alain Roy, UW-Madison and OSG; William Allcock, ANL

Large parallel computing systems are sufficiently complex that their optimization in a production computing setting presents formidable challenges. Building and operating a reliable computing instrument from millions of unreliable components is extremely hard and getting dramatically more so. The boundaries between the computing instrument and the datacenter itself are disappearing as the systems involved become more concurrent and tightly coupled. Because of this, most HPC centers and HPC vendors have entered into ambitious programs of system management that seek to organize and monitor the ways that such systems are run and how changes are implemented. System management software currently focuses on three major areas:

- Jobs (SLURM, Cobalt, PBS, Torque, Condor…)
- Node health and testing (INCA, RSV, NAGIOS, CACTI, Cerebro, Zenoss…)
- Change control (CFengine, BCFG2, Puppet, RPM…)

All three areas lie at the interface between stakeholders. Change control intersects center management and vendor responsibilities. Node health includes these two as well as users. Lastly, monitoring and provisioning jobs is the major interface between center managers and their customers/users.

Discussion and findings by topic

Jobs

Schedulers in use include Cobalt, Torque/MOAB, SLURM/ MOAB, PBS Pro, LSF, Condor, and SGE.

Best Practice: Use the same tools (like Torque/MOAB) across an entire site. Allow policies to differ between resources. Users and administrators benefit from the consistency.

Best Practice: Provide well-defined, well-considered, openly published APIs to your software so that behavior can be modified at sites.

Challenge: It's important to get good performance in job launching. Open-source software mixed with proprietary hardware makes this harder, unless you have the right interfaces.

New technology: How do we deal with a scientific workflow that needs to access computation, storage, network, etc.?

Challenge: Resiliency to failures gets more complex when we consider complete (long-running!) scientific workflows and increasing scale of systems.

Node Health and Monitoring

Challenge: There is greater diversity in monitoring software than scheduler software.

Challenge: It's hard to bring data together and correlate it across systems.

Best practice: Target your resources towards data management (federation), not data collection. Separate data from data collection. We have (and will always have) a wide variety of data collection systems—we need something common for data management/data formats.

Challenge: A barrier to big improvements is that we have something that works for today, and incremental improvements seem easier.

Challenge: Knowing what to watch, getting the right information, bringing it together in an actionable way.

Best practice: Maintain a historical database of failures, replacements and maintenance, and periodically validate outages.

Change Management

Tools in use: BCFG2, Cfengine, Puppet, OneSIS. (Fewer tools than monitoring, similar number of tools to scheduling. These are all open source tools, though some systems ship with proprietary tools.)

Best practice: Keep configuration in source-code repository (subversion, etc.) and treat it like software development. And keep it backed up.

Best practice: Make changes in one known spot, then push them out once they are right. Or a three-step process: test and development, qualification, then production. Have a test and development system and verify that changes happened, and monitor it systematically.

Best practice: Have a change protocol which is managed. Don't change during off-hours. Have a well-defined process for deciding what changes should be made to the system, and when they should be made. Example: Tuesday-Thursday are good days, during business hours, not Friday-Monday.

Cross-Cutting Issues

Challenge: Support models. There is a trade-off between commercialization (get someone else to do it cheaper) and open source (so we can do research and fix problems). Other models: Task order: pay company to implement it. Convince vendor it will be useful to them in the future. We need to manage diversity of our products. Sometimes there are too many overlapping tools, sometimes not enough diversity. There is a tension between varied needs and shared experience and increased efficiency.

Best Practice: Community building. We need a community so we can agree on what our problems are and what we're trying to accomplish, so we can decrease the diversity of systems. If we speak with one voice, we have more clout in the vendor (or open source) community.

Suggestion: We should periodically review the state of the union of our products. We need a forum in which to do this. Some product areas (jobs) are more mature, and might focus on reducing diversity. Some product areas (monitoring) are less mature, and may be more exploratory. A centralized server which acts as a clearinghouse for information about HPC software including status and performance information would see broad attention from the HPC community. Right now, no such resource exists.

Breakout Group 4: System Software

Group Leaders: Shane Canon, LBNL; Sue Kelly, SNL

Participants: Buddy Bland, ORNL; Robert Bohn, NCO/NITRD; Jeff Broughton, NERSC/LBNL; Bob Ciotti, NASA NAS Division; Mark Gary, LLNL; Gary Grider, LANL; Pam Hamilton, LLNL; Daniel Hitchcock, DOE/ASCR; Paul Iwanchuk, LANL; Janet Lebens, Cray Inc.; Tinu Ogunde, ASC/NNSA/DOE; Rob Pennington, NSF; Stephen Scott, ORNL; Krystyne Supplee, Cray Inc.; Vicky White, ORNL.

OS and I/O software for HPC systems run the gamut from open source (e.g. Linux and Lustre) to proprietary (e.g. Catamount and GPFS). The complexity also varies extensively from lightweight to full featured. This session explored the life cycle development and maintenance practices that are key to successful deployments of these software components.

Key findings:

1. There is a continued need for Alliance-style or Path Forward-style initiatives similar to those done within DOE/NNSA's Advanced Simulation & Computing (ASC) program. This includes matching the ASC level and scope of the problems with commensurate funding.
2. The current software stack breakdown won't sustain us to exascale. There are too many cross-cutting issues (e.g., resiliency and scaling).
3. Access to specialized test systems and large-scale systems is critical.

Session Process and Discussion

We began with introductions. Each attendee provided their name, organization, and role within the organization. Most attendees also offered their motivation for selecting this session and their top concern. The provided information is included in other parts of this session report. As part of the introduction, we also indicated if our system software was (a) vendor supported, (b) open-source downloaded, or (c) internally developed and

maintained. This information did not prove to be particularly illuminating as there was an even distribution of responses, with many sites doing all three.

The workshop cross-cutting questions were sufficiently comprehensive that they focused the discussion for the entire session. A summary of the discussion related to each question follows. The top four answers are bolded, but we provide the other responses that we discussed as well. We began with question 2 as it provided a natural flow to the other questions.

Breakout Question 2 (answered first): What are the top challenges?

Funding for system software is in allocations too small to have an impact for future HPC systems. As mentioned in the findings, the current software stack is likely not viable for exascale systems. Yet, funding levels force researchers to target only a portion of the existing software stack in their work. A cross-cutting effort is needed that addresses the entire problem and solution space.

Collaborations are difficult to establish given the existing funding distribution levels as well. Larger teams require more coordination, which requires more time, which unfortunately requires more funding. In order to maximize development time, external collaborations are minimized.

Lastly, funding for maintenance of system software is almost non-existent. Placing the code in the open source community does help spread out the burden. But even the contributors to open source products require funding to perform the maintenance. Some of the HPC-specific products (e.g., parallel file systems, schedulers and highly scalable network protocols) do not have the broad support community that products such as Linux enjoy. Therefore the maintenance burden rests with a small set of consumers.

Scalability concerns in HPC have been a recurring theme since massively parallel systems were introduced in the late 1980s. The introduction of multi-core processors has only exacerbated the scalability problem as more processors share access to memory and network resources. While it is important that the applications find the inherently parallel portions of their science, we must provide them with interfaces to the system software that are naturally expressive and complement their science focus. Additionally, system software must scale in its own right by using efficient/tuned algorithms and minimizing resource drain caused by factors such as OS noise and communication buffer overhead. These problems remain unsolved as core counts continue to increase and existing solutions become insufficient.

Resiliency: Hand in hand with scalability concerns are concerns with reliability and fault tolerance. The ever-growing number of components reduces the raw mean time between interrupt (MTBI). Effective resiliency techniques within the system software could improve the MTBI perceived by the application.

Complexity: In general, the low-hanging fruit to address scalability and resilience has been "picked clean" and future improvements will come with a complexity cost. Plus, there are additional issues that are gaining importance. Power management is reaching a high level of urgency — the current flop-per-watt curve is not sustainable to the exascale. System software needs to play a significant role in managing power consumption. Noisy operating systems impede the ability of a processor to quiesce to lower power states. Scheduling algorithms may need to take into account power needs as well. File systems may need to consider a hierarchy based on power consumption as well as access times and reliability.

System software research also needs to address the complex interoperability of system software components. Currently, it is rarely possible to change one component without having to take down the entire system.

What is the right balance between the above? What is proper frequency of course correction? These challenges, being phrased as questions, say it all. Funding, scalability, resilience, and complexity have to be balanced against each other as system software is designed, developed and deployed.

Gathering requirements is extremely difficult for HPC system software. Scientific applications developers often learn their programming skills on desktop UNIX-based systems. These systems support features (e.g. shared libraries, dynamic process creation, and run-time interpreted languages) that do not scale well, but significantly enhance programmer productivity. Additionally, application developers typically try to target their implementation for a wide spectrum of parallel systems and are often not interested in custom interfaces to enhance the high end of parallelism. The list of requirements is quite extensive and appears mutually exclusive with the HPC requirements of scalability, high resiliency, low complexity and high performance.

What is the right licensing model for system software? Again, the challenge articulated as a question describes it well. Based on the introductory survey, vendor proprietary and open source solutions are common in HPC facilities today. No current model is without issue.

Breakout Question 1 (answered second): What are the best practices and tools? Inside HPC? Outside HPC?

While there are many examples of successful approaches to developing system software, there were several high-level approaches that were identified as being particularly effective. Those were structured collaborations, PathForward-style initiatives, ASC Alliances and testing at scale.

Examples of **Structured Collaboration** include HPSS and OpenFabrics. One critical feature of these collaborations was the early procedure to establish a governance model that clearly defined how decisions were made for the software and the roles and responsibilities of the various players.

PathForward led to products like Lustre that likely would not have come about if left to simple market forces. The unique requirements of HPC and the limited market size make it difficult for vendors to make a business case to develop certain products. PathForward addressed that situation by identifying key challenges of the day and issuing contracts to fund development. Furthermore, the level of funding was commensurate with the scale of the problems to ensure some reasonable chance of success. The problems that PathForward dealt with were still fairly focused on addressing a specific problem. Current day initiatives would need to tackle large, cross-cutting challenges like fault tolerance and scaling.

The **ASC Alliances** tackled hard problems, but also strove to build up centers of excellence around key areas. This had the benefit of creating a pipeline of talent that extended beyond on lifetime of the initial contract work. The most successful alliances had a high level of engagement between the program managers, the institutes funded to do the work and the end-users. The close oversight ensured that the projects stayed on target and that the funds were leading to output that would provide real value.

Testing at Scale continues to be a critical factor to success. While it is not feasible for vendors to operate large-scale systems to support development and testing, many of the most challenging issues often only arise at scale. There are several successful approaches that have been employed to address this challenge.

- Community-oriented test systems such as Hyperion at LLNL provide vendors and developers with access to a system significantly larger than what they can typically access. Hyperion provides access and time based on the level of contribution to the system.
- Dedicated system time (DST) is often used to verify and test new software releases prior to placing them in production. However, sites like NERSC and ORNL have provided additional time to vendors like Cray to test software releases that carry additional risks. This was one component that was not well addressed in the PathForward projects.
- Decommissioned systems could potentially offer a platform for testing and development. Many academic sites may find an older system useful for this purpose. However, the infrastructure and facilities requirements may exceed the capabilities of many academic sites.
- For more novel systems, simply having full access to a small test system can be important. Many system software developers and researchers cannot afford to acquire and operate even small versions of systems like Cray XT and IBM BlueGene. While most sites have test and development systems, those systems are often limited to center staff and are needed to support the production systems.

In addition to these high-level best practices, other approaches and practices were discussed.

A Policy for Sustainability is needed to ensure that software developed with funding from DOE considers the entire lifecycle of the software. Too often software is developed following an ad hoc approach. While this can be effective in producing initially useable software, the process often breaks down as the software transitions from development to the maintenance stage. Having requirements and policies in place that require looking at the entire lifecycle could improve the situation.

Commonality in the software stack is preferred by the user community. The DOE/NSSA tri-labs have united to use a Common Computing Environment (CCE) for their ASC-related capacity computing. Both the hardware and software are largely common across the CCE systems in the tri-labs. The users find the consistency to be a productivity gain in their work. ASC high-end capability systems continue to push the envelope of performance and require more custom software to scale to the system peak.

Breakout Question 3: What new technologies are needed?

The group identified five areas requiring technological advances:

- System software suitable for heterogenous computing
- Exploiting virtualization in HPC
- Machine diagnostics (self organizing and self diagnosis)
- System software to provide other sw layers information to be fault tolerant
- Power aware software

In general, the HPC system software stacks assume a relatively homogeneous set of compute processors. To increase performance, accelerators may well become the ubiquitous solution in the HPC systems of the coming decade. These heterogeneous compute nodes are not adequately supported by the system software. Virtualization support in CPU hardware is another feature not adequately utilized by today's HPC system software. Virtualization has the potential to extend the life of existing applications that assume specific hardware and/or system software configurations.

Crossing the hardware and software boundary are machine diagnostics that more accurately identify an existing or potential system problem. Increasing machine complexity makes it harder and harder for text-based logs and ad-hoc diagnostics to pin point a problem. A natural follow-on to these improved machine diagnostics would be a standard software interface to the information. A common API that could be exposed to the application would allow the application to decide the best course of action, given its current processing state.

Another useful new API would be one that exposes power settings to system and application software. Recent research has shown the power states can be reduced during some processing, such as network communication without any deleterious effects. A slight increase in run time can be offset by a significant reduction in power utilization.

Day 2: Software Stages
Chair: Susan Coghlan, ANL

Breakout Group 5: Software Planning (Strategic management, procurement, funding)

Group leaders: Mark Gary, LLNL; and Craig Tull, LBNL

Group 5 Participants: William Allcock, ANL; Shane Canon, NERSC (note taker); Bradly Comes, DoD HPCMP; Sudip Dosanjh, SNL; Mark Gary, LLNL); Daniel Hitchcock, DOE ASCR;, Patricia Kovatch, NICS/UTK; William Kramer, NCSA/University of Illinois; Viraj Paropkari, NERSC; Rob Pennington, NSF; Stephen Scott, ORNL; Yukiko Sekine, DOE ASCR; Henry Tufo, NCAR; Craig Tull, LBNL (lead).

HPC software products often have lives that span multiple decades while serving many generations of machines and operating environments. Careful project planning is the foundation upon which these projects are built. From requirements gathering and cost estimation to collaboration and team building, deliberate and realistic planning is the key to product usefulness and longevity. But how do HPC software projects differ from typical software development projects? Do HPC requirements or the HPC community introduce impediments to successful planning? Successful collaboration? Are we in the HPC community successfully leveraging non-HPC methodologies? This session will address these questions, investigate the facets of good software planning, and explore alternative planning approaches.

- Measuring use and forecasting needs
- Assessing criticality and establishing support priorities
- Identifying and managing cost
- Strategies for interagency and international cooperation
- Strategic choices in software licenses

Discussion and findings

Planning software for HPC systems inherits its most significant challenges from the fast-paced and quickly changing directions in computer system architecture and scale. The inability to forecast the nature or size of next-generation computing systems is coupled with the uncertainty in what demand for compute and data resources will come from the scientific community. Planning software for uncertain architectures and uncertain applications is best dealt with through frequent discussion and incremental approaches to testing plans on what equipment is available. Determining and scheduling costs is additionally important given the above uncertainties.

Breakout Question 1: "What is unique about HPC planning?"

Though there are significant differences, there are also many similarities between industry and HPC planning. Many industries (e.g., Google and Sun) do far-reaching software research and share many of the challenges of out-year unknowns with HPC centers. However, HPC software planning involves integrating extant products and green-field products, and HPC centers are forced to support a wide spectrum of software, much

of it being developed out of research projects. This often leads to software which is in very active use while still being in very active development. Commercial software vendors have the advantage of being able to constrain their scope as necessary to both minimize risk and areas of expertise.

The most obvious and fundamental difference between HPC and industry is that of funding sources and reward models. Commercial software typically has a very well defined support/sustainability funding model identified in advance, something that is often absent in HPC environments. In industry, quantitative measures of success (namely profit/loss) are more definitive and obvious. HPC centers typically struggle with the questions of when research becomes development and when development becomes sustainment, while commercial software efforts typically have much more explicit and better managed phases.

Breakout Question 2: "What rules exist for HPC project planning?"
Session attendees are rarely impacted by explicit mandates surrounding planning. While orders such as DOE Order 414.1C exist, these mandates typically incorporate risk-grading (considered to be very important) which allows them to be adapted to the level of planning rigor appropriate to a project. CMMI methodologies have been used in the context of particular software projects (e.g., HPSS). ITIL has been considered by ASCR but has never been mandated, as it applies more appropriately to operations rather than software. Some sites (e.g., LLNL) have Software Quality Assurance programs that provide templates to guide project planning activities. ASCR requires open source software licenses for government-funded products (with a slight preference for BSD) and encourages the copyrighting of software.

Breakout Question 3: "What are your best practices?"
A focus on early investment in research, infrastructures and people is critical to HPC success. Closely managed, well funded and supported Laboratory LDRD investment, ASC PathForward and ASC Alliance-like programs must be included in, and influence, HPC planning as they are absolutely critical to success. Session members brought up the need for a category of seed funding at low dollar levels, allowing spot funding of quick-turnaround research, since even LDRD-funded work seldom allows exploratory research. Investment in test infrastructures (particularly at-scale test infrastructures such as Hyperion) was repeatedly identified as being crucial. Focused attention on balanced I/O and support infrastructures is a common denominator in successful HPC environments.

HPC centers make successful use of a wide variety of planning vehicles. ASCR relies upon annual Lehman reviews of HPC sites. While these reviews typically focus on hardware and facilities, software projects are covered because of their tight coupling with all aspects of center success. Centers also use daily, weekly, quarterly and yearly meetings along with reports and blueprints as planning mechanisms that link hardware and software efforts. These efforts depend heavily on requirements gathering (a best practice/challenge discussed in its own section below). Some sites make use of agile/extreme programming methodologies and find these techniques very effective at getting valuable early feedback from tightly coupled users.

The use of metrics and quantitative tracking techniques to inform planning/investment decisions by developers, facilities, and agencies was singled out as a critical best practice. Some projects have successfully risked backlash by installing "spyware" to monitor and categorize software usage to benefit planning and requirements gathering. Informed planners are then able to use measured innovation approaches leveraging industry standards and open source software while targeting effort where standards and products are deficient or non-existent.

Positive and collaborative vendor relationships are common to HPC success stories. A number of different models exist and have been characterized in an IESP (International Exascale Software Project) Workshop:

- Funded Investigation
- Fully defined purchase
- Design and development
- Co-design and co-development
- Base + value add

A discussion of these models is detailed in the IESP report:
http://www.exascale.org/mediawiki/images/6/6f/Paris-full-report.pdf

Breakout Question 4: "How is project cost planned, managed, monitored?"
HPC sites integrate project cost planning and monitoring into their periodic review mechanisms. Earned Value Management (EVM) techniques, while appropriate for very large integrated hardware/software projects, are too rigid for the vast majority of software technology projects and have been abandoned when attempted. Limiting the lifetime and scope of projects was identified as a strategy used by some organizations to avoid ballooning project costs and feature creep. Continual feedback and verification of software's relevance and appropriateness is necessary.

Breakout Question 5: "When software development funding ends, what are your strategies to maintain it?"
Unlike commercial software, HPC community software rarely has a pay-for-use model and it is frighteningly common for viable HPC products and projects to find themselves without funding for sustainment. This is a major challenge identified by most workshop participants. Agencies must be willing to not only support sustainment funding, but to plan for it in advance, at the inception of a project. In the planning stage it must be realized that the personnel developing a product may not be the appropriate individuals charged with maintenance. Attention to providing adequate rewards for those maintaining/sustaining products is very important.

The idea of an HPC software sustainability center was discussed. Such a center would support HPC tools and software stacks for the greater HPC community. While this concept would have a number of serious operational, political and funding hurdles to clear, it was identified as a concept worthy of a thought exercise. Such a center could provide common tools and techniques for long-term software sustainability and provide a

coherent view of HPC community software. An HPC software sustainability center could be a virtual community of individuals and institutions sharing a common goal and some common tools and resources, or it could be one or more brick-and-mortar group whose main goal is to serve the wider HPC community.

Breakout Question 6: "What strategies should be used for interagency and international cooperation?"

There are many examples of successful interagency cooperation in support of HPC, including but not limited to: HPSS, Lustre, the Open Science Grid, OpenMPI, and the ASC Tri-Lab software stack. Each of these examples had different governance models and mechanisms for collaboration, but they prove that large HPC challenges can be conquered through interagency and international collaboration. As exascale power costs lead to fewer large HPC facilities, most centers will find themselves invested in a remote hardware and software collaborative domain.

Collaboration does present many challenges. Trust relationships take a long time to build and are vital when achieving collaborator buy-in during the planning stage. How one supports collaboration between potential competitors with proprietary technologies and techniques is a challenge. Legal, cyber security, licensing, export control, and foreign national security issues are all hurdles commonly encountered in large-scale interagency and international collaborations.

Breakout Question 7: "How do you gather requirements and how accurate are they?"

HPC software project success hinges on accurately identifying requirements early in the planning stage and continually verifying them throughout a project's lifetime. A wide variety of techniques are used to gather HPC requirements with the common theme of project planners meeting one-on-one with customers — whether that be in multi-disciplinary focused customer visits, workshops, or imbedding developers with customer teams. These meetings should embody constraintless thinking and use case discussions teamed with end-user education and negotiation to help mitigate unrealizable expectations. Tracking requirements in a database that is regularly reviewed and vetted is a best practice.

Requirement accuracy and validation is always a challenge. The continuous use of use metrics, trouble ticket history, and feature request analysis is important when forming requirements. Requirements should always be stated in a fashion that is objectively measurable. Agile development strategies benefit from alpha user feedback loops enabling requirement vetting and establishing realistic expectations on the part of both users and developers.

Breakout Question 8: "What other top challenges surround HPC planning?"

A significant challenge surrounding HPC planning is that planning typically occurs at the component level. Cross-cutting requirements and design principles spanning the entirety of the software infrastructure are typically absent. This causes a myriad of problems and inefficiencies that are barely tolerable at current scales and will become untenable at

larger scales (e.g., the exascale). Resilience is an excellent example: While individual components may implement resiliency techniques, they often do not communicate much-needed failure information to surrounding components, making full system resilience, which will be critical in exascale architectures, very difficult to achieve.

HPC centers face a myriad of technology and vendor unknowns. Even mid-range planning faces questions surrounding hardware architecture and the availability and performance of algorithms operating at scales never before demonstrated. These factors greatly increase project risk and contribute to project delays. Funding agencies must realize the speculative nature of HPC efforts and appreciate that, as in most research areas, false starts will be made and dead-ends encountered.

Along with the aforementioned challenge of sustainability funding, HPC projects often have difficulty bridging the funding gap between research and development and product use by domain scientists. A successful research project may well produce a valuable product, but may lack any follow-on planning or funding. Because of the speculative nature of research, planners and funding agents should provide a mechanism for transitioning research products into applied tools. Such a mechanism must incorporate a reward mechanism for software sustainability beyond the count of the peer-reviewed publications appropriate for research.

Breakout Question 9: "What new technologies are needed to help with HPC planning?"
Any technology that helps to enable collaboration is applicable to HPC planning. Technical industry standards, such as common portable data formats, provide seed points for planning collaborative HPC software products. However, no explicit new technologies were identified by workshop participants.

Breakout Group 6: Software Development

Group Leaders: Deb Agarwal, LBNL; Paul Iwanchuk, LANL

Software engineering best practices typically include a thorough regimen of testing, bug tracking, documentation and release. Software design practices such as agile development and continuous integration are widely employed in developing code. An HPC environment brings with it several unique aspects including that development of software for HPC systems is often concurrent with the maturation of the target system. HPC software includes the applications, the libraries, the operating system as well as software targeted to testing the software and hardware environment. Validation and verification play a central role along with regression testing, tracking and documenting results. Similarly, there is a concerted effort to assure key applications are ready for the new architecture.

This session will focus on the life cycle development and maintenance practices that are key to successful deployment and operation of these software components. We will follow software practices in the maturation of a typical HPC system from procurement

through production to end of life. This session will address best practices at these stages, rather than addressing software components. Questions such as: How do you assure production readiness? What is your reliance on in-house development vs vendor support? Is your custom environment helping or hindering end use? What is your ability to use other DOE institutional resources to complete your mission? What are the barriers? DSTs and updates? What role is fault tolerance and resilience playing in your future?

- Testing, tracking (results, bugs, dependencies)
- Continuous integration, agile
- Validation and verification

Discussion and findings

Best Practices

- **Waterfall models do not work in HPC** at or near the application level – agile programming has become the de facto practice with continuous integration (except OS and file system in some cases). Test-driven development has become standard. Tight development loop with:
    - Requirements
    - Development and documentation
    - Evaluation
    - Test
    - Deploy to early users and get feedback
    - Repeat above

- **Design for minimized maintenance and functional success**
    - Work with hardware vendors for early testing at vendor's and customer's site of libraries, software, etc. API design collaboration
    - Test-driven software design and development
    - Thorough testing
        - Unit testing of code
        - Functional testing
        - System testing
        - Integration testing
        - All testing at scale (when possible)
    - Release criteria (synthetic workload tests)
    - Dedicated system time for tests at scale

- **Application-level testing of functionality during development and acceptance**
    - End-to-end testing using real codes lies less. Instrument codes to track usage and performance
    - Identification of gaps and direction of funding toward addressing technology gaps (e.g., Lustre file system resulted from this)

- **Communicate and adapt**

- Tiger teams (cross functional multi-disciplinary teams and including vendors)
  - During the life-cycle of the system to solve either focused or end-to-end problems and address issues
  - Working with vendor in ways that minimize "risk of engagement and responsibility"
- Issue tracking on all systems – using systems like TRAC and Jira
  - Includes hooks into underlying code repositories
  - E-mail developer directly and start an issue

Challenges

- Agile software programming model
  - Software scope creep versus valuable features – how do we differentiate?
  - Hard to put together a detailed schedule and budget for development
- Getting a holistic view to understand interactions between layers and repercussions of changes. Methodology for continually test to check functioning end-to-end. (e.g. library has its own build/test environment and each package has its own).
- Retiring software and systems – old versions and end-of-life reasons
  - Who is responsible for compatibility?
  - Process for roll out of new versions
  - New versions of xxx – causing recompile of all apps and libraries
- Testing at scale and with end-to-end applications is important, but keeping this test suite up-to-date is difficult.
- Vendors don't necessarily have the perspective of the users and more interaction during development is needed
- Managing the different executables and compilers hard when everything has to be recompiled
- Funding does not have a holistic view. No explicit funding for migrating applications to new systems.
  - Dealing with new programming models, new architectures and scales, new libraries, new design paradigms …
  - Hard to predict what will be coming – current practice is to study the range of expected architectures
  - Changes for future disruptive architectures often cause failures on existing architectures
  - What is the evolutionary path?
- Predicting when a new architecture will be deployed in machines – when should the software start migrating to the new architecture?
- Getting more insight into architecture at the chip level to handle resource contention and related issues

New Technologies

- System resilience – need a holistic approach to the system resilience as a whole (hardware and software)
- Methods for diagnosing problems at scale better
    - Ability to triage and debug
    - Visualization tools for debugging
    - Better summary data about how the system is behaving
- Many core – programming models for dealing with this transition
- Simulators or emulators for systems at scale
    - Using tools to predict performance on new architectures – need better tools

Breakout Group 7: Software Integration

Group Leaders: Pam Hamilton, LLNL; Vicky White, ORNL

Fielding an HPC system requires a huge deployment of system and security software, networking infrastructure, a development environment and user applications, inevitably from different sources. We, the DOE lab customer, are responsible for putting these pieces together and making them work. How do you work with the vendor to refine your requirements and then verify they are met through system acceptance testing? Especially when some of the work may be done at the vendor facility and then on site? Key issues:

- Modularity, interoperability
- Packaging, distribution
- System stabilization: co-development with vendors between system assembly and production use

Discussion and Findings

General Findings

- People and relationship issues are at least as important as technical challenges.
- We are doing okay on issue tracking.
    - Sites use a variety of tools: TRAC, RT, RightNowWeb, Bugzilla, Remedy, Frontrange. All can be effective.
    - Vendors (at least Cray) often allow customers to see their own bugs and public bugs in the vendor's tracking database.

Best Practices

- **Promote famliarity with and a culture of using version control tools and processes.**
    - Version control impacts all levels of the HPC process. Tools are needed to track changes and impacts.
        - We need to keep track of version dependencies.
        - We need to keep track of the versions and conditions under which faults occur. This can help pinpoint what changed when the fault occurred: operating system, library, compiler?
    - Users get bitten by new versions of tools that break their software.
    - Common sources of versioning headaches include:

        - Software that is embedded in user applications source
        - Proprietary software which builds in specific versions
    - The impact of mis-versioning:
        - Failed runs
        - User time spent working (writing) around software mismatches
        - Loss of confidence by users to adopt software for fear it will soon break
    - Solutions include SVN, CVS, CleaCase, Metronome (NMI), Eclipse.
        - Something is better than nothing; adapt choices to site and staff.

- **Adopt testing frameworks and practices.**
    - Integration is ongoing, so testing must be as well.
        - HPC systems are complex and dynamic enough that assumptions about function must be tested. The assumption that "once tested all is well" does not hold.
    - Failure to test turns users into testers (loss of productivity).
    - Use dedicated systems or dedicated time slots for invasive testing.
    - Let friendly/early users contribute to tests.
    - Parameterize your tests to solve the combinatoric problems of having 5 shells, 4 compilers, 32/64 bit, and 10 test cases. Users will explore this space, so your tests must as well.
    - Take advantage of the fact that different sites use different tests and/or test approaches. This allows more problems to be uncovered than if each used the same methodologies.
    - Even though each site may not use the same set of tests, all tests and tools should be available to all sites.
- **Allow users to contribute to integration.**
    - Early in a HPC project hold face-to-face meetings of the different teams necessary to field the system: OS support, networking, security, applications support, user assistance.
    - Continue regular team meetings, even if they cannot be face-to-face.
    - Have an entire user applications team whose purpose is to teach users how to make the best use of the machines and to help them port their codes. Communicate the state of the system and its software. Sometimes system outages or changes are interpreted as software problems.
    - Connect early access to systems being integrated to user contribution to the shake-out process. Don't charge if possible.
    - Assign a support person to meet regularly with each user group as a cross-matrixed member of their team. Have the support person try out the user group's code and get it to compile on the new system.
    - Without user support, many users try a new system once, and if it doesn't work, they walk away. Hand holding helps.
    - Hold workshops for training users and helping them port their codes.

- **Build strong vendor relationships**.
    - It is hard to overestimate the importance of this step.

- HPC systems are too large to set up at the vendor site for testing, and the large-scale integration most often must happen at the customer site.
- Give the vendor time and flexibility to do the setup and early testing at the site.
- Arrange to have vendor personnel temporarily or permanently assigned to your site.
- Give the vendor representative code which he can test at his site. Even if his test site is smaller than yours, many bugs can be flushed out early this way.

Breakout Group 8: Sustaining Software

Group Leaders: Charles Bacon, ANL; David Montoya, LANL

This session will identify approaches and efforts to better provide a sustainable environment for the software we use. HPC software comes from a mix of commercial vendors, open-source communities, and in-house development. Regardless of the source, if the support structure for the software disappears, the users are faced with serious consequences. As software and tool developers, what models do we use to interact withthe open-source communities to help maintain a long-term support structure? How does a group's support model need to change over time as software moves from a development/research state to one that runs in a production environment? What are the concerns and issues from organizations that are responsible for providing a stable production environment? What approaches have organizations used for sustainability, and how can the user community participate in a way that leads to greater sustainability?

- Support models, upgrade paths
- From prototype to facility software
- What is open-source support?

Discussion and findings

Finding long term stakeholders in software provides one path to charting direction for sustained support of software. Insofar as the costs of providing enduring support can be shared across those stakeholders, sustainment is made easier. HPC has some great examples of software which has made the transition from research prototype to a solid production computing package. There are many other packages which, though valuable, do not find long term support outside of an initial outlay of R&D funding.

Open-source communities are a double-edged sword when it comes to software support. It can be difficult to plan the level of support that will be obtained, its duration, and also changes in direction or focus as developers modify the software. Software supported in this way may reach a stable supported state, but exactly the opposite may happen as well, and much depends on the sociology and communication with open source developers.

Deciding which software is worth sustaining and prioritizing effort around those choices is a difficult aspect of sustaining software.

Best Practices/Tools: Inside HPC

- Separate prototype development from production development
    - Sustainable software that can absorb good research ideas. Make this a process. (VisIt/MPICH/ACTS Collection)
    - Testing at scale — schedule the time because vendors/open-source communities don't have the resources
    - Risk analysis – identify consequences of failure (vendor based, other)
    - Include software capability and support in the system RFP (PAPI, IEEE floating point ...) to garner vendor support
    - Multi-organization contracts – can we expand, what is the future model as we include other organizations (HPSS, Open|SpeedShop)

- Inventory software, fund according to weighted importance to lab
- Sustainment reviews should be done by independent eye
- Funding models

Best Practices: Outside HPC

- Open-source groups that have a good quality gatekeeper
- There are model open source communites
- External vendors have separate organizations that support tools

Education/Training

- Letting other know capabilities, expanding user community
- Gather from user community – what should change to engage and what should continue to be sustained – feedback loop

Worst practices

- Wacky licensing: Use standard open source licenses. Also provides a backup plan.
- Unhappy success: Management doesn't want to pay for support of external users.
- Duplicate support contacts at a small scale, often duplicating scope. To avoid, look at HPSS funding model.
- Prototypes that are not ready to move toward production.

Top challenges?

- Cross-cutting: providing hardware for testing at scale
- Market may drive vendors and open-source communities away from HPC requirements
- Prioritizing software sustainment is hard
    - Research funding doesn't pay for sustainment
    - Since software support funding is categorized under facilities – it competes with hardware purchases
    - New technologies required?
- Facilities funding for software
- Cross-organizational coordination on process (inventory of strategic tool needs, coordination of support)

- Single external organization to coordinate funding
- Is a facility approach needed to build support structure for tools? (virtual software community)
- Software dependencies – how do we better manage tests that can provide this information?

Conclusion

This third workshop on HPC best practices promoted a great deal of discussion on wide-ranging topics. The central trend in most of these discussions is an acknowledgement that the time is right to turn our enginering attention, which has been so productive applied to *computer systems*, to HPC *software systems*. The ecosystem of HPC software is a complex system which must allow for the unfettered ability to creatively develop new algorithms and applications, but simultaneously requires rock-solid stability for production computing. Such a diverse set of requirements yields a broad set of best practices whose applicability depends on context and center needs. However, a few common threads run trough all those topics, which if acted upon, would float all boats in the HPC software ecosystem. Attention to software engineering and taking a long view of the software lifecycle can best be made tangible through the following actions by HPC centers and managers:

1. **Recognize that software is an enduring facilities concern.**
2. **Develop an online catalog of available HPC software which describes its use, current state and stakeholders.**
3. **Provide allocations of computer time and improved access to test systems to make software perform better once it goes into production.**

The HPC science community is increasingly benefitting from and dependent upon shared software stacks. The time is right to bolster our attention to software engineering.

A strong sentiment from workshop attendees has to do with the degree to which HPC and data center work is reactive in the short term. Today's best practices focus in part on eduring rules for HPC, but are largely formed around today's concerns — massive concurrency now being chief among those challenges. Continued discussion and a process to revise and reprioritize best practices would be valuable through future workshops and meetings.

The Workshop Series

This was the third in a series of workshops sponsored by DOE ASCR and NNSA. The first, focusing onPetascale Systems Integration into Large Scale Facilties, was held May 15-16, 2007. The second workshop, Risk Management Techniques and Practice Workshop for High-Performance Computing Centers, was held Sept. 17-18, 2008 in San Francisco. Of the 21 attendees of the 2009 workshop completing a survey, 19 of them stated they would benefit from a fourth workshop in 2010. The top topics suggested were resiliency, user support, application scalability, and power consumption.

List of 2009 Attendees

Deb Agarwal (LBNL)
Dong Ahn (LLNL)
William Allcock (ALCF)
Kenneth Alvin (SNL)
Chris Atwood (HPCMPO/CREATE)
Charles Bacon (ANL)
Bryan Biegel (NASA Ames)
Buddy Bland (ORNL)
Brett Bode (NCSA)
Robert Bohn (NCO/NITRD)
Jim Brandt (SNL)
Jeff Broughton (NERSC)
Shane Canon (NERSC)
Shreyas Cholia (NERSC)
Bob Ciotti (NASA NAS)
Susan Coghlan (ANL)
Bradley Comes (DoD HPCMP)
Steve Cotter (ESnet)
Nathan DeBardeleben (LANL)
Sudip Dosanjh (SNL)
Tony Drummond (LBNL)
Mark Gary (LLNL)
Ann Gentile (SNL)
Gary Grider (LANL)
Pam Hamilton (LLNL)
Andrew Hanushevsky (SLAC)
Daniel Hitchcock (DOE ASCR)
Paul Iwanchuk (LANL)
Gary Jung (LBNL)
Suzanne Kelly (SNL)
Patricia Kovatch (NICS/UTK)
William Kramer (NCSA)
Manojkumar Krishnan (PNNL)
Janet Lebens (Cray Inc.)
Sander Lee (NNSA)
Lie-Quan Lee (SLAC)
Ewing Lusk (ANL)
Osni Marques (DOE ASCR)
David Montoya (LANL)
Krishna Muriki (LBNL)
Tinu Ogunde (ASC/NNSA/DOE)
Viraj Paropkari (NERSC)
Rob Pennington (NSF)
Terri Quinn (LLNL)
Randal Rheinheimer (LANL)
Alain Roy (Univ. Wisconsin-Madison/OSG)
Stephen Scott (ORNL)
Yukiko Sekine (DOE ASCR)
David Skinner (LBNL)
Rebecca Springmeyer (LLNL)
Jon Stearley (SNL)
Krystyne Supplee (Cray Inc.)
Henry Tufo (NCAR)
Craig Tull (LBNL)
Andrew Uselton (NERSC)
Daniela Ushizima (LBNL)
Tammy Welcome (LBNL)
Vicky White (ORNL)
Dean Williams (LLNL)
Kathy Yelick (NERSC)
Mary Zosel (LLNL)

3rd Workshop on HPC Best Practices: Software Lifecycles

SURVEY RESULTS: A total of 35 surveys were returned. For each of the first two questions, participants were instructed to allocate 15 tokens among the items according to importance. In the below tables, "Tokens" is the total number of tokens that item received, and "People" is the number of people who allocated any tokens to that item. Items have been sorted below according to decreasing total tokens (most important first).

Top Best Practices and Tools (15 tokens per person)

Tokens	People	Item
99	27	Software testing (unit/coverage/regression/functional/scaling) that is automated, broad in scope, and extended to communities beyond developers
58	21	Join or form structured collaborations (HPSS consortium, OpenFabrics, etc)
57	19	Establish collaborative relationships with vendors
56	22	Well-considered, well-defined, published APIs for all software layers
52	18	A forum for periodic review the state of the union of our software products
45	18	Establish well-defined release criteria, formal issue tracking, milestone planning
40	16	Maximize ease of installation and use / invest in excellent packaging
32	25	Include SW capability and support in the system RFP (PAPI, IEEE floating point, etc) to garner vendor support
24	12	Develop robust, portable, focused libraries and APIs to underpin tool development
21	11	Agile/Extreme programming is very effective when feedback from tightly-coupled users is available
18	9	Change management for system management
12	7	Resist feature-creep / stay focused on key goals and stakeholders
4	2	Target system management resources towards data management (federation), not data collection

Top Challenges and Technology Needs (15 tokens per person)

Tokens	People	Item
85	23	Funding program priorities which place sole emphasis on new research preclude software hardening and sustainment
72	21	System software to provide other SW layers information to be fault tolerant
66	20	Cross-cutting challenges in resiliency and scaling to current software stacks
48	15	Access to specialized test systems and large scale, use of older

		machines for test
47	16	Libraries to insulate applications from new languages and lower-level libraries needed to take advantage of emerging hardware
38	15	No central clearinghouse for HPC software needs, resources, dependencies, use cases
36	12	Deriving machine diagnostics from varied streams of data
30	9	System software suitable for hetero computing
26	9	Impediments to achieving widespread adoption of automated software testing
22	12	Energy/Power aware software
19	5	Market may drive vendors and open-source communities away from HPC requirements
17	7	Exploiting virtualization in HPC
7	5	Predicting when software development should start based on emerging architectures and bets upon them

Questions from Breakouts

1. Would you see value in development of a general build/test framework for HPC libraries that would promote best practices and help establish a common infrastructure for software testing?

30	YES
1	NO

2. In your opinion, what would add more value to the HPC applications community:

24	Promoting compatibility and interoperability, via standards and API definition, between HPC libraries, particularly those that facilitate parallel data management, communication and future programming paradigms; or
7	Developing new libraries that provide a general software layer to largely insulate integrated applications from changes and complexity in parallel programming models?

For questions 3 through 5, attendees were instructed to rank items. For scoring, it was assumed that 1 indicated highest importance, so the items below have been sorted by decreasing total score (most important first).

3. Rank the value of the following identified best practices in HPC system software.

84	Forming relationships with vendors and open source providers through contracts and active participation in the community
86	Providing test beds (for specialized systems), access to large platforms, perhaps through open resources available to the entire community, and dedicated test shots on production systems.
96	Structured collaborations, such as those employed by HPSS, OpenFabrics, and NNSA/ASC's PathForward and University Alliances program
104	Having a policy for sustainability

4. Rank the importance of the following identified challenges in HPC system software.

88	Scalability
91	Resiliency
94	Complexity (many interacting components, problem isolation, interoperability of components etc)
104	Funding is in too small of increments to have impact and to allow collaboration

5. Rank the urgency of the following missing technologies from HPC system software.

72	System software that provides information to other software layers in order to be fault tolerant
96	System software suitable for heterogeneous computing
117	Machine diagnostics (self organizing and self diagnosing)

141 Power aware software
151 Exploiting virtualization in HPC

6. How valuable would a simulation/emulation capability for new architectures at scale be to you? Scale of 1-4 with 4 being the most important

20 System with applications
17 System
15 Chip

7. Should a cross-facility committee be formed to identify technology gaps and sustainability issues to identify priorities for accelerated development and ideally impact funding (e.g. PathForward)?

31 Yes
0 No

8. What is your strategy for addressing the change to programming models for many-core?

14 we are preparing for it and are shovel ready
10 we have formed a committee but no results yet
2 we are ignoring it
1 the vendors and underlying libraries will handle this seamlessly

Feedback

Participants were invited to provide written feedback on the following questions. Below is a summary of the responses.

Do you have any feedback (positive or negative) about the workshop?

- (6) Well run/organized. (5) Good discussion & coverage. (4) Good overall. (4) Good location.
- Good connections to individuals working on problems.
- Good mix of full groups and breakouts.
- On target.
- Good forum – breakout session discussions are as valuable as the documented outcome.
- It was great to bring multiple experts together to identify best practices and challenges important to centers. It is not always easy to find center-focused workshops.
- There was a lot of latitude given to the breakout sessions. It made it a little confusing to start, but allowed for more creative ideas to foster.
- There's a tension between thinking aloud, exploring, building relationships, and driving toward a final report. Don't let the latter dominate the former!

- (3) The first day was too long!
- I'm unclear of the long-term impacts.
- Weighted too heavily towards HPC platform technology community (weak on application/libraries).

- Poorly organized breakouts. Little consistency in approach or results.
- The breakouts should [have] also involved vendors. Presentations from each participant. (Leaning Negative)

How could this workshop have been improved?

- (3) More info and guidance prior to event so participants can better prepare.
- (2) Wider representation (less ASCI focused/more DOE labs, vendors, software providers)
- More balance between applications (integrated end-user) and platform OS tools.
- HW issues interposed themselves too often. SW comes in many types – a taxonomy at the beginning would help.
- It may be useful to split up discussion of consumers and produces of software since it confused things a little.
- Concrete recommendation with risk mitigation plan.
- Little more time nailing down path forward.
- Not interested in dinner SW audit list.

- More time to prepare breakout summary. Longer break before dinner.
- More discussion time on 2nd day.
- Better mics, better visibility of screens.
- A firm agenda. Particular tools discussion.
- Insufficient time in some sessions.

What would be productive follow-up action items for this workshop?

- (4) Written report.
- Develop action plans for (2) sustainability including funding approach, (2) top workshop findings, (1) facilities for testing at scale and diverse environments.
- (2) Info-sharing mechanisms for ongoing discussion (wiki, mail list).
- (3) Sample "Path Forward" efforts in this area.

- Follow-up from ASCR on how many workshop results were used in planning at the DOE level.
- Update software inventory.
- Meet as a BOF at SC09.

Would you like to see a 4th HPC Best Practices Workshop next year (Y/N) and what topic(s) would you like to see addressed?

Yes (19):

- (2) Resiliency
- User support
- Best practices in structuring large HPC procurements
- Testing.
- Power consumption – ways to manage power.
- Systems-tools-infrastructure (software)
- Simulators for processors, including interconnect
- Data intensive science, knowledge-discovery and science
- Many more; scalability
- Open source software

- Whatever the topic, pay attention to dates and if in late Sept – please avoid Yom Kippur.
- Let's drop "best practices" and have something that tries to identify gaps & barriers.
- Something that would attract more applications specialists and/or end users – this would facilitate a healthy exchange between the providers and the consumers of HPC.

No (2)

Yes/No not circled:

- I wouldn't mind a year off – however resiliency of systems is an urgent issue.

Below is a transcription of the raw responses.

Do you have any feedback (positive or negative) about the workshop?

- Excellent. Good discussion and planning. Good connections to individuals working on problems.
- Great location. Good mix of full groups and breakouts.
- Good coverage and organization.
- Very good discussions, well organized, great location, weighted too heavily towards HPC platform technology community (weak on application/libraries).
- Good discussions.
- Good overall.
- (2) Good job.
- The logistics were great (hotel, food).
- Very good – full days but covered info well.
- Well run, good location.
- Well run, on target, organized to obtain results.
- Good forum – breakout session discussions are as valuable as the documented outcome.
- It was great to bring multiple experts together to identify best practices and challenges important to centers. It is not always easy to find center-focused workshops.
- There was a lot of latitude given to the breakout sessions. It made it a little confusing to start, but allowed for more creative ideas to foster.
- I'm unclear of the long-term impacts.
- There's a tension between thinking aloud, exploring, building relationships, and driving toward a final report. Don't let the latter dominate the former!
- Poorly organized breakouts. Little consistency in approach or results.
- The first day was long!
- The first day could have been squeezed, the breakouts should [have] also involved vendors. Presentations from each participant. (Leaning Negative)

How could this workshop have been improved?

- Little more time. Nailing down path forward.
- Better set of background materials for people to see background info.
- Would be nice to have a longer break before dinner, if only to get some exercise.
- More time to prepare breakout summary.
- Not interested in dinner SW audit list.
- Insufficient time in some sessions – adjust. Distribute work items before meeting.
- HW issues interposed themselves too often. SW comes in many types – a taxonomy at the beginning would help.
- More balance between applications (integrated end-user) and platform OS tools.
- Concrete recommendation with risk mitigation plan.
- Less ASCI focused, more inclusive/aware of other DOE labs.
- The time allocated to discussion on the 2nd day could be increased.
- It may be useful to split up discussion of consumers and produces of software since it confused things a little.

- Perhaps more info/guidance prior to meeting so that we can come in more prepared w/ local site info and some content prepared.
- Better mics, better visibility of screens.
- A firm agenda. Particular tools discussion.
- More participants from outside HPC centers would have been appreciated. There were a few vendor representatives but I would appreciate experts from other vendors and key software providers.

What would be productive follow-up action items for this workshop?

- After the workshop notes are put together – what are the plans and tasks?
- A joint funding proposal to actually address some of the issues. For example – get someone to fund a repository for logs and info addressed.
- Establish a common site for sharing [??][1] and documents.
- Act on the top best practice by token vote.
- Develop action plans for: 1) facilities for testing at scale/diverse environments, 2) new "path forward" programs, 3) fund sustainability.
- Follow-up from ASCR on how many workshop results were used in planning at the DOE level.
- Update software inventory – and have discussion – at least DOE-wide (OS+NNSA) about software sustainment approach that is coordinated.
- A wiki, possibly requiring logins, where an ongoing discussion could be hosted would facilitate an ongoing discussion.
- Ensure written report is completed and made available.
- Top 3 recommendations.
- Meet as a BOF at SC09.
- What would some potential path forward efforts look like?
- (2) White paper.
- Do [survey question 7], like HEC_FSIO for all HPC.
- Look at which items are of most importance to most attendees & form groups to pursue possible actions/solutions.

Would you like to see a 4th HPC Best Practices Workshop next year (Y/N) and what topic(s) would you like to see addressed?

Yes (19):

- Simulators for processors at full interconnect [??]
- (2) Resiliency
- User support
- Open source software
- Data intensive science, knowledge-discovery and science
- Many more; scalability
- Best practices in structuring large HPC procurements
- Whatever the topic, pay attention to dates and if in late Sept – please avoid Yom Kippur.
- Testing.

[1] [??] is used to indicate a word that the transcriptionist can not read.

Software Survey Results

The below table originated from the International Exascale Software Project (IESP). Workshop participants were given a hardcopy of the table and asked to provide additional data about software used at their sites, using the following key:

1	Must have
2	Most apps need
3	Most developers want
-	Don't use
R	Risk mitigation

The table contained 316 data points from three sites prior to the workshop. As a result of the workshop, the table contains 576 data points from 11 sites - a significant increase in knowledge regarding software usage at HPC centers.

Application Support	NICS	LANL	Blue Waters	DoD Create	ANL	BSC-CNS	LLNL	NERSC	ORNL	SNL	Unknown
netCDF, parallel I/O library used by some apps	1	3	1		2	3	1	1		3	1
ACML AMD Advanced Core Math Library		3			-		2	1		1	-
ATLAS					-	2	3	1			3
BLACS Use vendor version when possible	1							2		2	3
BLAS		3	1		1	1	3	1			1
BLAS (vendor) Use vendor version when possible	1		1				3	1		1	1
CDO "CDO is a collection of command line Operators to manipulate and analyse Climate model Data. Supported data formats are GRIB, netCDF, SERVICE, EXTRA and IEG. There are more than 400 operators available."					-	3					2
CFITSIO CFITSIO is a library of C and Fortran subroutines for reading and writing data files in FITS (Flexible Image Transport System) data format. CFITSIO provides simple high-level routines for reading and writing FITS files that insulate the programmer from the internal complexities of the FITS format. CFITSIO also provides many advanced features for manipulating and filtering the information in FITS files.					-	3		-			-
DAP Open-source Project for a Network Data Access Protocol					-	3		-			1
ESMF					3	3		-			2
ESSL			1		2	2		2	2		1
FFTW GPL	1	3	1		2	2		1	1	2	3
FTP Used to access storage			1		-		1	1		2	-
FTPS Used to access storage			1		-	2					3
GMP						3					-
GOTO					2	2				3	3
GSL The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers					3	3					-
HDF4	1		3		-	3		1		2	3

HDF5		1		1		2	2		1	1	2	3
HDF5_PAR	I/O library used by some apps			1		2		1	2		2	3
HDF5_SERIAL	I/O library used by some apps			1		2		1	2		2	3
HTAR	Small file aggregation for archiving	1	2	1		3		1	1		3	3
Hopper	Optimized File Movement with GUI			1		-		1	2		2	-
IOAPI							3		-			-
JASPER	Jpeg 2000 library						3					-
LAPACK		1	3	1		1	1	3				3
LAPACK (vendor)	Use vendor version when possible			1		1		3			1	3
LP_SOLVE							3					-
MKL	Intel Math Kernel Library					3		3	2		1	1
MUMPS							3					-
Metis	Used by several apps				1	3	3	3			2	3
Secure FTP					1							
Trilinos											2	

Application Support

	NICS	LANL	Blue Waters	DoD Create	ANL	BSC-CNS	LLNL	NERSC	ORNL	SNL	Unknown
NCDAP						3					-
NETCDF			1		2	2	1	1		3	1
PARMS					3	3					-
PFTP	Parallel ftp used to access storage	3			-		1			1	-
PNG	Portable Network Graphics				-	3					3
PORT						3					-
ParMetis	Used by several apps				3	3	3	2	2		3
PetSC	Math library used by some apps	1		1	1	3	3	1	1		-
SCALAPACK		1	3			1	1	2		2	3
SLEPC						3		2			-
SPARSEKIT						3					-
SPRNG		1		2		3					-
Scalapack	Use vendor version when possible	1	3	1				2		2	-
WSMP						3					-
netCDF	I/O library used by some apps	1		1		3		1	2		3	1
run/proxy	Need equivalent code steering ability						3				3	-

Programming Environment

shells (tcsh, ksh,sh, bash)	User scripts in many shell languages		3	1	1	2		1	1		3	3
ADA						-	2					-
ANT				3	3	3	3					-
AUTOCONF			3	3	2	2	3	3			1	2
AUTOMAKE			3			2	2	2	3		1	2
BIOPERL	Bioperl is a collection of Perl modules that facilitate the development of Perl scripts for bioinformatics applications.						3					-

BOOST		3		2		2				1	-
Blockbuster high-resolution movie player							3				-
C/C++		2	1	1	1	2	1	1		1	3
C/C++/FORTRAN95 "F95, Cray pointers, OpenMP in v4.1"		2	1	1	1		1	1		2	1
CAF Co-Array Fortran			1		3			2			R
CEPBATOOLS The set of performance tools including instrumentation tools as well as visualization tools such as Paraver and Dimemas. Paraver is a very powerful performance visualization and analysis tool based on traces that can be used to analyse any information that is expressed on its input trace format. Dimemas is a simulation tool for the parametric analysis of the behaviour of message-passing applications on a configurable parallel platform.						3					-
CMAKE CMake is a cross-platform system for build automation				1	1	3	1	2		1	-
CPPUNIT CPPUnit is a unit testing framework module for C++				2		3	2				-
Programming Environment	**NICS**	**LANL**	**Blue Waters**	**DoD Create**	**ANL**	**BSC-CNS**	**LLNL**	**NERSC**	**ORNL**	**SNL**	Unknown
DDT Competing parallel debugger					-	3	3	1			-
DynInst Dynamic code instrumentation		3			-	3	3	2			-
Eclipse			1	1	-	3		2			-
Ensight Visual debugging and visualization		2					3			3	-
FERRET						3					-
Fortran 77/90/95		3	1	1	1	1	1	2		1	1
GASnet communications layer for messaging			2					2			2
GCC		3	3	2	1	1	1	1		1	3
GIMP image editor					-		3				3
GLPK						3	3				-
GNU libc			3	3	3	3	2	1		3	3
GNUPLOT		3	3		2	3		3		3	3
GRACE		3				3					-
GRID-S						3					3
GTK+					1	3					3
HPM Limited functionality on BG/P			1		3		2				3
IHPCT					3	3	3				-
IMAGEMAGICK		3			1	3					-
IOTRACK											2
IPM scalable performance profiling	1		1					1			-
Intel Thread Checker thread correctness tool					2		3			2	3
Java (JRE & SDK)		3		3	2		3	2		3	-
LIBTOOL					3	2	3	3		3	3
LaunchMON Co-location of tool deamons with apps							3	2			3
MOLDEN						3					3
MPICH-MX	1				1	1					3
MPICH2-MX					-	1					-

MPIP					-	3	2				-
MRNet	A Multicast/Reduction Network	3			3		3	2			1
Make	Used by most apps	3	1	1	1	1	1	1		1	1
NCAR	graphics program library				-		3				-
NCARG					-	3					-
NCL						3					1
NCVIEW						3					1
OTF	"Open Trace Format (scalable, open)"	3					3			3	1
OpenGL	graphics programming library				1		1	2			3
OpenMP	Preferred threading model	1			3		2	1		3	3
OpenlSpeedShop	Transitioning from SGI to ?	3					3			3	-
PAPI	Used by many tools and applications	1	1	2	2	2	2	1		2	1
PARAVIEW					3	3					1
PAVE						3				1	3
PERFMINER	Adapted to Moab+SLURM system queue and improved for scalability					3					3
PHP		3			3	3		2			-
POV-Ray	ray tracer						2				-
POVRAY						3					-
OpenMPI				1						3	-
Jumpshot					1						-
pNetCDF					1						-
VL3					3						
MPI (vendor)					1						

Programming Environment

		NICS	LANL	Blue Waters	DoD Create	ANL	BSC-CNS	LLNL	NERSC	ORNL	SNL	Unknown
ParaView	data analysis and vis tool					2		2	2		1	-
Perl	Script interface for several apps	1	2	1		3	2	1	1		1	2
Python	Script interface for several key apps	1	2		1	3	1	1	1		1	1
QT4	License required for commercial use				1		3	3				-
Qt	License required for commercial use					1		3	1		3	-
R		1				-	3	3			3	3
RasMol	molecular visualization tool							2				-
SBML							3					-
SCALASCA							3					-
STAT	Scalable Stack Trace Analysis							3	2			-
Scalasca	scalable trace analysis tool										3	-
StackWalker	Portable call stack walker							3				-
TCL				1	2	3	2	1	1			2
Tau	Used regularly by several key apps				2	2		3	1		3	-
ThreadSpotter	OpenMP application tuning tool							3			3	-
Tool Gear	"GUI for valgrind, mpiP, others"							3				-
Totalview	Primary parallel debugger used	2	1		3	1	1	1	1		1	3
Totalview Memory Tools	Key for memory use at scale issues		3			-		1	1		2	3
Totalview Reverse Debugging	Step backward during debugging!					-		2	1		3	3

UPC	Universal Parallel C			1		3	3	3	2			-
VIS5D							3					-
VMD						3	3					-
VTK-5.0.0						3	2				2	3
Valgrind	memory correctness tool		3			3	2	1	2			3
Valgrind Helgrind	thread correctness tool							3	2		2	3
Valgrind Memcheck	memory correctness tool							1	2		2	3
Vampir	Vampir provides an easy to use analysis framework which enables developers to quickly display program behavior at any level of detailampir provides an easy to use analysis framework which enables developers to quickly display program behavior at any level of detail			3			3	3	1			-
Vampir/VampirServer	vampir next generation (Vis mpi trace)			3				3			3	-
VampirTrace	"Flexible MPI,OpenMP,HPM tracing library built on the OTF library"			3				3			3	-
VisIt	"80% Visual Debugging, 20% Movies"			1		2	3	1	2			-
WXGTK							3					-
X11	Many apps and tools use X11		1	1		1		1	1		1	3
apprentice2									1		2	-
cvs				1		3		3	1		3	3
dotkit	Enhanced module support		3					2				3
emacs	IDE for some users		3		3	3		1	1		3	3
gcov	Code coverage important for V&V		3			3		3	2			-
git					3	3		1	3		3	-
globus	grid services					3		2	2			1
gprof			3		3	3	2	1	2		3	3
ldd	Shared library usage reports needed		3	2	3	3		1	2		3	3
subversion (svn)					2							
Programming Environment		**NICS**	**LANL**	**Blue Waters**	**DoD Create**	**ANL**	**BSC-CNS**	**LLNL**	**NERSC**	**ORNL**	**SNL**	Unknown
memP	Parallel memory heap profiling							3				-
modules		1	3			-		3	2		3	3
mpiP	Used for both app and system tuning							1			2	-
mplayer	movie player					3		3				-
pyMPI	Python MPI extension				3	3	3	1				3
secure VNC	"Need secure, fast, remote X display "			1	3	-		3				3
shells (bash)	User scripts in many shell languages		3	1	3	1	2	2	3		1	2
shells (tcsh)	User scripts in many shell languages		3	1	3	1	2	2	3		1	2
svn			3	3	1	2		3	3		3	3
tcl/tk	GUI for many small tools		3	1	2	3		3	3		3	3
vim	"Handles huge files, unlike vi"			1	2	1	2	2	1		2	3
vmd	molecular dynamics visualization tool		3			2		2				-
xxdiff/tkdiff/meld	Graphical file diff tool					-		3				-
softenv						1						

Sys Software

MPI, MPT Based on MPICH2 version 1.0.2	1		1	1	1		1	3		1	1
PowerMan, FreeIPMI Power Management					1		2	1		3	-
genders, dist Configuration Management Tools					-		3			3	-
lm_sensors, FreeIPMI HW Sensors Support					1		3	3		3	-
lxbios, cmos_util CMOS/BIOS utilities					3		3			3	3
whatsup, skummee, cerebro Host Monitoring					-		3				1
yaci, gedi cluster installation					1		3				3
ANL UserBase					S						-
ARMS ANL RAS Management Suite					S						-
Asset Tracker Breakfix Asset Tracking					-					1	-
CHAOS See below for RAS components					-		1			3	-
CSA											-
Cacti Monitoring / Security					S			2			3
Catamount					-			1		1	-
Cbench Scalable Benchmarking and Testing Framework					-					1	-
Cobalt-Accounting					S						1
Cobalt-ResMangr					S						1
Cobalt-Scheduler					S						1
ConMan Console Management					S		2	1		3	3
DIM						1					-
EDAC Hardware Error Detection/Correction							1			3	-
GPFS system wide filesystem			1		1	1	3	1			1

Sys Software

	NICS	LANL	Blue Waters	DoD Create	ANL	BSC-CNS	LLNL	NERSC	ORNL	SNL	Unknown
Ganglia Adapted to MareNostrum in order to improve scalability and include other fetaures not supported			1		-	1		1			1
IBM CNK					1		1				1
IBM ION					S		1				-
LDAP System wide auth and unix groups	1	1	1	3	S		1	1			3
LMT Lustre Monitoring Tool					-		3	1		3	3
Linux	1			1	1		1	1		1	1
Lustre Forwards requests to Lustre servers on service nodes					-		1	1		1	1
MOAB Tri-lab Workload Manager	1	1			-		1	1			-
MPI "IB, MVAPICH, MVAPICH2 based from Ohio State"		2	1	2	-		1	1		1	1
MPI-IO Part of MPICH2-based MPI from Cray	1		1		1		2	1		2	-
MX					1	1					-
Moab Tri-lab Workload Manager		1			-	1	1	1	2	2	3
Munge Scaleable Cluster Authentication					-		3			3	3
NFS "Support included in Linux 2.6 kernel distribution; NFS v3 support, not sure about NFSv4"		1	1	1	1	1	1	1		1	1
Nagios Monitoring / Security	1				S	1	3	2			1
OFED/OpenIB		1			1		1	1		1	1

OpenSsh (with OTP extensions)	system access					1		1	1		1	1
PBS Pro					3	-			1	2	2	2
PVFS	Parallel file system					1			1			2
Pam	Used widely for authentication					1		1	1		1	1
Portals	Based on Portals from Sandia					-			1		1	-
PowerMan	Power Management					1		3				3
RASilience	Breakfix Asset Tracking										1	-
RT			3			S		1			2	-
Red Hat Linux	CHAOS built on top of Red Hat								1		1	1
Request Tracker	Breakfix Asset Tracking					-					1	-
SHMEM	Part of xt-mpt			1		-			1		2	2
SLES					3	1	1		1		1	1
SLURM	Highly Scalable Resource Manager		3			-	1	1			1	R
SUSE	on BG/L login and service nodes					1		3				1
Sun HPC Stack	Tools and configurations for Sun cluster management					-					1	R
Torque		1	1			-			1		2	R
Trac						3			1		1	-
ZeptoOS-CN						R			2			R
ZeptoOS-ION						R, S						R
cfengine	Configuration Management Tools		3	1		-		3	1			-
crms	"Have a module file, but references non-existent directory"									2	2	-
gPXE	Boot-over-IB Support for PXE Boot				3						1	1
ldap			1	1		1		1			1	1
mrsh	Scalable rsh implementation					-		3			3	-
oneSIS	Diskless image management system					-					1	-
pam				1		S		1			1	3
pdsh	Parallel Shell					S		3	3		1	-
ssh				1		1		1	1			1
yaci	cluster installation		1		2	-		3				-
jira					1							
Romio						1						
PXE						S						
Zenoss			1									
OpenMPI			1									
puppet		1										