Post-Workshop report for the Third International Workshop on Software Engineering for High Performance Computing Applications (SE-HPC 07)

Jeffrey Carver
Department of Computer Science and Engineering
Mississippi State University
carver@cse.msstate.edu

Abstract

This is the report from a one-day workshop that took place on Saturday, May 26, 2007 as part of the International Conference on Software Engineering in Minneapolis, MN, USA.

Background and Statistics

High performance computing (HPC) systems are used to develop software for wide variety of domains including nuclear physics, crash simulation, satellite data processing, fluid dynamics, climate modeling, bioinformatics, and financial modeling. The TOP500 website (http://www.top500.org/) lists the top 500 high performance computing systems along with their specifications and owners. The diversity of government, scientific, and commercial organizations present on this list illustrates the growing prevalence and impact of HPC applications on modern society.

Recent initiatives in the HPC community, such as the DARPA High Productivity Computing Systems program, recognize that dramatic increases in low-level benchmarks of processor speed and memory access times do not necessarily translate into high-level increases in actual development productivity. While the machines are getting faster, the developer effort required to fully exploit these advances can be prohibitive. There is an emerging movement within the HPC community to define new ways of measuring HPC systems, ways which take into account not only the low-level hardware components, but the higher-level productivity costs associated with producing usable HPC applications. This movement creates an opportunity for the software engineering community to apply our techniques and knowledge to a new and important application domain.

Furthermore, the design, implementation, development, and maintenance of HPC software systems can differ in significant ways from the systems and development processes more typically studied by the software engineering community:

- The requirements often include conformance to sophisticated mathematical models. Therefore, the requirements may take the form of an executable model in a system such as Matlab, with the implementation involving porting to proper platform.
- Often these projects are exploring unknown science making it difficult to determine a concrete set of requirements *a prioiri*.
- The software development process, or "workflow" for HPC application development may differ profoundly from traditional software engineering processes. For example, one scientific computing workflow, dubbed the "lone researcher", involves a single scientist developing a system to test a hypothesis. Once the system runs correctly once and returns its results, the scientist has no further need of the system. This approach contrasts with more typical software engineering lifecycle models, in which the useful life of the software is expected to begin, not end, after the first correct execution.
- "Usability" in the context of HPCS application development may revolve around optimization to the machine architecture so that computations complete in a reasonable amount of time. The effort and resources involved in such optimization may exceed initial development of the algorithm.

This workshop provided a unique opportunity for software engineering researchers to interact with researchers and practitioners from the HPC application community. Position papers were selected from researchers representing both communities. The consensus among the workshop attendees was that the overall quality of these papers was quite high, due in part to the lack of other venues to report this type of work. These researchers shared their perspectives and presented findings from research and practice that were relevant to HPC application development. A significant portion of the workshop was also devoted to discussion of the position papers with the goal of generating a research agenda to improve tools, techniques, and experimental methods for HPC software engineering in the future.

To lay a proper foundation, and provided valuable input throughout the data, three invited speakers from the HPC community provided important information on software engineering challenges from the HPC perspective and ideas for future research. These invited talked prompted some interesting discussion and highlighted challenges for the future.

The list of attendees at the workshop included: Rola S. Alameh (*University of Maryland*), Edward B. Allen (*Mississippi State University*), Jeffrey C. Carver (*Mississippi State University*), Mikhail Chalabine (*Linkoping University*), Ian Gorton (*Pacific Northwest National Laboratory*), Christine Halverson (*IBM*), Lulu He (*Mississippi State University*), Michael A. Heroux (*Sandia National Laboratories*), Lorin M. Hochstein (*University of Nebraska*), Jeffrey K. Hollingsworth (*University of Maryland*), David Hudak (*Ohio Supercomputing Center*), Andrew Johnson (*Honeywell*), Jeremy Kepner (*Lincoln Laboratory*), Frederick M. Lowe (*Los Alamos National Laboratory*), Michael O. McCracken (*University of California, San Diego*), José Muñoz (*National Science Foundation*), Tien N. Nguyen (*Iowa State University*), Victor Prankratius (*University of Karlsruhe*), Adam Porter (*University of Maryland*), Atanas Rountev (*Ohio State University*), and Richard Vuduc (*Lawrence Livermore National Laboratory*).

Presentations

This section provides a brief synopsis of each presentation, along with any follow-up discussion. All of the papers and presentations are available on the website of the workshop

(http://www.cse.msstate.edu/~SEHPC07/).

Keynote - José Muñoz – "The NSF CI Vision and the Office of CyberInfrastructure"

In the keynote presentation, José Muñoz explained how the interest of the National Science Foundation in Cyberinfrastructure was related to research on software engineering for HPC applications. The Office of CyberInfrastructre (www.nsf.gov/oci) has the stated mission of "greatly enhance the ability of the NSF community to create, provision, and use the comprehensive cyberinfrastructure essential to 21st century advances in science and engineering." He first explained three important activities that must be performed in harmony: 1) Transformative Application of CyberInfrastructure to enhance discovery and learning, 2) Provisioning to create and deploy advanced CyberInfrastructure, and 3) R&D to enhance the technical and social effectiveness of CyberInfrastructure environments. Then he highlighted some opportunities with the National Science Foundation for researchers to pursue funding related to these activities. Relevant programs include (while some of these solicitations may have already closed for the current competition, the information is still useful in preparation for future competitions):

- Strategic Technologies for CyberInfrastructure (PD 06-7231);
- Accelerating Discovery in Science and Engineering through Petascale Simulations and Analysis (NSF 07-559)
- High-End Computing University Research Activity (HECURA)
- Community Based Data Interoperability Network (NSF 07-565)
- Engineering Virtual Organizations (NSF 07-558)
- CI-TEAM
- Software Development for CyberInfrastructure (NSF 07-503)

Ian Gorton – "A High Performance Event Service for HPC Applications"

This presentation described work conducted by Ian Gorton, Daniel Chavarria, Manoj Krishnan, and Jarek Nieplocha at Pacific Northwest Laboratory on the event service portion of the Common Component Architecture (CCA). Gorton, et al., implemented the CCA event service using a traditional software architecture approach: publish-and-subscribe. The goal of this work was to build this higher-level messaging interface atop lower-level message passing approaches like MPI, with minimal performance penalty. Two case studies were presented to highlight the successes and shortcomings of the approach and note room for improvement [3].

Richard Vuduc – "Tool support for inspecting the code quality of HPC apps"

This presentation described work conducted by Thomas Panas Dan Quinlan, and Richard Vuduc at Lawrence Livermore National Laboratory, on a tool for visualizing the structure of HPC codes and computing metrics. This research is based on the premise that software development in the HPC environment is generally done in an ad hoc manner (i.e. it does not follow standard software engineering processes). Even so, developers need to be able to easily obtain information about the quality of their code during development. This paper described a tool that allows developers to visualize relationships among code elements (e.g. call graph, file-include graph) using the metaphor of a city to reduce the complexity of the visualizations. Applying the tool to some standard benchmark applications showed that interesting information could be gathered that may not have been as obvious when using more standard approaches [6].

Rola Alameh – "Performance Measurement of Novice HPC Programmers' Code"

This presentation described work conducted by Rola Alameh, Nico Zazworka and Jeffrey K. Hollingsworth at the University of Maryland on performance analysis of student HPC codes. They report on a series of classroom studies to understand how novices develop software for high performance computers. To collect data, a series of automated tools were created called the Automated Performance Measurement System (AMPS). Using AMPS, they were able to gather a large amount of data to pose two interesting hypotheses. First, "spending more effort does not always result in increased performance for novices." Second, "the use of higher level MPI functions promises better performance for novices [1]."

Michael O. McCracken – "Measuring & Modeling HPC User Productivity: Whole-Experiment Turnaround Time

This presentation described work conducted by Michael O. McCracken, Nicole Walter, and Allan Snavely at the University of California, San Diego and the San Diego Supercomputer Center on providing decision-support to scientists for improving turnaround time. They discuss a problematic trend that existing measure of productivity (i.e. FLOPs) are not providing adequate insight into the real bottlenecks experienced by scientists. They propose an approach for eliciting workflow information from scientists and building workflow model simulations, which can be executed to answer various "what-if" questions when balancing trade-offs in planning their code execution [5].

Christine Halverson – "Was that Thinking?"

This invited presentation provided a perspective on the measurement of programmer productivity from a social scientist working with IBM. IBM has conducted a series of productivity studies using both automatically collected data and observational data. An important, and difficult, issue is finding the right balance between the two types of data to provide the necessary insight into the activities being studied. One interesting question, that prompted the title of the presentation, is: when the automatically collected data indicates that the user was idle, were they "thinking" about how to solve the problem, or were they taking some type of a break? The main issues raised during this presentation focused on a challenge to researchers to gain a better understanding of what it is they are really trying to measure, and of the accuracy of the methods being used to perform the measurement. A concluding question that researchers in this area must consider is: "Can we build studies that combine automated and observational data and determine patterns of behavior to better make inferences?"

Jeremy Kepner – "Quantitative Productivity Measurements in an HPC Environment"

This invited presentation discussed work performed by Jeremy Kepner, Bob Bond, Andy Funk, Andy McCabe, Julie Mullen, and Albert Reuther at MIT's Lincoln Laboratory on assessing the productivity of HPC systems. The discussion focused on how to define and measure productivity, Using Lincoln Lab's LLGrid

system as an illustrative case study. These researchers define productivity as "utility over cost." Using this information, a Return on Investment figure can be calculated to better understand the value that an HPC center is getting from its supercomputer. The cost variable includes multiple constituent parts: 1) time to parallelize a code, 2) time to train the users, 3) time to launch the code on the supercomputer, 4) time to administer the supercomputer, and 5) cost of the system. Kepner suggested that LLGrid's Matlab-based, interactive HPC system has dramatically increased usage and productivity over the C/Fortran-based batch queued systems commonly found at other HPC centers.

David Hudak – "Developing a Computational Science IDE for HPC"

This presentation described work performed by David Hudak, Neil Ludban, Vijay Gadepally, and Ashok Krishnamurthy from the Ohio Supercomputing Center on the benefits developers obtain by using integrated development environments (IDEs) instead of a collection of unrelated tools. The needs of HPC developers require a different type of IDE than traditional software developers. In particular, HPC developers need to perform remote, interactive services. Some of the challenges in designing a successful IDE are the result of the observation that the HPC developers often do not consider themselves to be programmers. So, while the concept of an IDE is appealing to this community, the implementation still needs refinement [4].

Michael A. Heroux – "The Trillinos Software Lifecycle Model"

This presentation described work performed by James M. Willenbring, Michael A. Heroux, and Robert T. Heaphy from Sandia National Laboratories on a proposed lifecycle model for HPC libraries. This work was motivated by the observation that while a lot of work was being done on projects that could be considered similar, very little reuse or coordination was occurring among them. As a result, the Trillinos lifecycle was developed to facilitate the design, development, integration and support of mathematical solver libraries. Because no single development model can address all of the needs of these developers, the Trillinos project is an approach that provides the flexibility to allow projects to move among different levels of maturity, each requiring different amounts of software engineering rigor. The concepts of software quality assurance and software quality engineering are important and integral at all stages of the process. A notable aspect of this lifecycle is an initial "Research" phase, which has no equivalent in traditional software engineering lifecycle models [7].

Discussion

After the presentations, a short discussion session followed that focused on the question: "How is Software Engineering in a research environment different from Software Engineering in a more traditional environment?" This question was motivated by a reoccurring theme that appeared during the earlier presentations. The members of the HPC community do not see value in many of the traditional software engineering concepts. Further discussion indicated that much of the reason for this different view of software engineering had to do with the motivation for writing software. Thus, it was important to further discuss the effects of writing software in a research environment. The starting point for this discussion, and one of the main contributions, was to define the difference between a "research" environment and a more traditional environment. There were two main types of differences discussed: differences in the overall plan and differences in the people involved. Finally, there was a discussion of the potential similarities between research environments and a subset of the more traditional environments. Each of these topics is discussed in more detail in the sub-sections that follow.

Research Plan vs. Business Plan

In research projects the teams tend to have a "research plan" as opposed to a "business plan", which a more traditional project would have. In a business plan, the focus is normally on how to make the best use of the available resources, including technical personnel like software engineers, to be financially successful. The decisions related to planning tasks and allocating personnel to those tasks are all driven by this underlying goal. Conversely, in a research plan, the focus is on obtaining new knowledge that will benefit the larger scientific community. Therefore, the process drivers may be quite different from those that would be derived from a business plan. In a research plan, the goal is discovery of new knowledge, so it is to be expected that requirements or even the scope of the project will evolve as more knowledge is gained. This flexibility of requirements may not be so common or viewed as positively in cases where the process is driven by a business plan. Finally, research plans account for the fact that research projects are inherently more risky than other types of projects. By definition, research is the investigation of something unknown, so there is always the risk that the software project could completely fail due to reasons external to the software itself. Projects that are driven by a business plan do not tend to face these same types of risks.

Personnel differences

The discussion suggested that different types of people are involved in HPC projects than in more traditional software development projects. It is common for the developers of HPC software to also be the users of that software. This situation is less common in other domains like information technology. The implication of this situation is that developers may not feel the need to use good software engineering principles because they know that if a problem arises during software use, they can just fix the problem. A second people-related problem is that people who are highly knowledgeable in the domain are usually not the same people who are experienced, and trained, software engineers. This situation results from the common belief that it is easier to teach software development to domain experts (i.e. scientists and engineers) than it is to teach the complex domain concepts to a software engineer.

Similarities between Research Environment and Traditional Environments

During the discussion, the focus shifted to trying to determine what subset of more traditional software engineering projects may be similar to research projects. This portion of the discussion posed more questions than it answered, which fed into the Research Agenda described in the Summary. The first idea was that certain types of internal software projects may be similar to research projects. Internal projects are those that are developed solely to be used in-house and not to be sold. Some of the similarities between these types of projects and research projects include: 1) planning may be more like a research plan than a business plan as

the requirements may shift often based on the needs of the organization; and 2) the user base will likely be made up of those that that are also developers of the software rather than external users. Another area where similarity may be found is in the area of risk. One interesting question that arose during the discussion was whether there are any groups of developers that are using the traditional software engineering methods to write high-risk software. This environment in which this software is written should be similar to the environment needed for research projects. In this case, a high-risk project is one in which the developers are unsure, a priori, if the requirements are feasible, tractable or even possible.

Breakout Groups

After listening to the presentations and discussion, the last activity in the workshop was to divide up into breakout groups to further discuss the issues. The goal of the breakout group session was to distill the information heard throughout the day into some concrete recommendations that could feed into a research agenda. Because the workshop participants came from two distinct backgrounds, software engineering and high performance computing, two breakout groups were created using this division. Each group was provided with a series of questions to address. The session concluded with a plenary discussion where each breakout group presented their results. The goal was to understand the similarities and differences in the views of the researchers from the two groups and arrive at a research agenda for the future. The results of each groups' discussion are presented in the following sub-sections.

High Performance Computing Group

The High Performance Computing group consisted of Christine Halverson, Michael Heroux, David Houdak, Jeremy Kepner, and Michael McCracken. This group addressed three questions as described below.

What are some software engineering techniques that have worked in the past?

The group identified a number if techniques that have been successful. While presenting this information, additional points were added by the entire group during the discussion. The first two topics identified were Performance Risk Analysis and Source Management. These two topics encouraged little discussion.

The group agreed that there are a lot of things that the software engineering community has produced that are practical and useful. The HPC developers would like these practices to be viewed like a buffet, where they can take what they would like and leave the rest behind. An example of the type of practices that are easy to pick and choose what works as well and are fairly easy to embrace are the Agile methods. On the other hand, this buffet approach is counter to the recommendations made by Kent Beck in his book on eXtreme Programming (XP). He believes, although it is only a hypothesis, that while some benefit can be gained by using only some of the individual practices, the majority of the benefit of XP comes when all the practices are used together [2].

For example, pair-programming has been useful in some situations. The HPC developers have not, and likely will not, adopt it universally, but it has been useful for training new developers. Also, when working on a very complex portion of the software, HPC developers have found pair-programming to be very useful. A second agile practice that has found some acceptance is the test-first approach. Anecdotal evidence from the workshop attendees indicated that once HPC developers adopt this practice, it is difficult to get them to give it up. Conversely, one member of the group reported some difficulties with motivating software engineering undergraduate students to use the test-first approach on their projects. Other agile methods that were mentioned as promising and well-suited to the HPC domain are: tight customer interaction and highly technical programming.

Another approach that has been beneficial is the creation of frameworks that abstract away the platform-specific information (the parallel machine). These frameworks have been more successful when they were domain-specific. Finally, the traditional, proven software engineering technique of code reviews were found to be helpful.

What are some things the HPC community does not need from software engineers?

There were some lessons learned from development for computational grids that motivated the list of items that are not needed by the HPC community. First, the idea of a BDUF (big design upfront) is not a good fit for the nature of the HPC domain. The BDUF approach does not work well if the core technical risks have not been mitigated. In addition, doing the software engineering correctly (e.g. requirements, OO design, …) can be worse than just being useless in the face of design changes. This situation is one of the drivers for leaning towards agile methodologies. Full-blown lifecycle models were also seen as problematic, because the developers, and customers, are not willing to wait long enough for these processes to complete. Furthermore, the funding for most of these projects comes from the government, who wants to be able to clearly track progress and see how the spending directly translates into functionality. This mindset makes the use of heavyweight processes difficult and unlikely.

What do you most need from software engineering researchers?

The experience of the HPC community with software engineering principles has been mixed. On the one hand, there have been some extremely successful large HPC projects that had not adopted identifiable SE practices. On the other hand, they recognize that failure to adopt good SE principles does hinder development. One member of the group told an anecdote of a computational scientist who needed help improving the performance of a finite-element code. However, the code was so poorly structured that the HPC consultants could not understand it, and therefore could not help the scientist.

The HPC group identified a set of high-priority items that they would like from software engineering researchers. First, they suggested a number of process and method improvements. Performance has to be influential in the design process. It is important for software engineers to realize, and develop methods, that help HPC developers design for performance from the beginning. The considerations of performance must come before those of functionality, because it is difficult or impossible to retrofit the software for performance. HPC developers also need help from software engineers when it comes to software architecture. The general practice in HPC development is to come up with a first version of the architecture that is too simple, followed by a second version that is too complex, followed finally by a third version that is just right. Another frustration faced by HPC developers is that they are re-

quired by managers to use standard software engineering lifecycle models, even when they do not fit in their environment. The HPC developers would really value some "expert testimony" from software engineering experts to support the argument they must make to their managers that many of these lifecycle models really do not fit the HPC domain. For example, HPC projects are often required to follow CMM guidelines when the projects do not match well to the requirements for such a process. The newer CMMi has helped with this problem, but it is still an issue.

Second, a set of tools was enumerated. In general, tools were requested to accommodate lightweight documentation, correctness testing, and aid in design software for testability. Those tools should also be designed to be used by scientists rather than software engineers. Examples of such tools can be found on the Sourceforge website. The Eclipse development environment also has some of these tools, but the consensus was that it was too heavy to be usable in many HPC settings. There was also the view that the Matlab debugger and editor were too heavy. They provide an interface to an enormous backend, so it feels like trying to "pull information through a soda straw". One last issue, is that many of these tools are designed for PCs and Windows, rather than Unix/Linux environments in which many of these developers work.

Finally, HPC researchers wish that when working with HPC developers software engineers would follow the processes they promote. For example, many from the HPC domain had experienced the situation in which a software engineer arrives with what they believe to be the solution/approach/tool/method that will save the day. The only problem is that often that software engineer has not invested the time to first collect the requirements of the system they trying to help (to identify what the real problem is and what solutions may not be feasible) before designing the solution. If software engineers would spend more time listening to HPC developers and understanding their real problems and the constraints of their development environments, they can likely arrive at better recommendations.

Software Engineering Group

The Software Engineering group consisted of Rola Alameh, Edward Allen, Jeffrey Carver, Mikhail Chalabine, Lulu He, and Lorin Hochstein. This group was addressed three questions as described below. Following-up on the earlier discussion differentiating research projects from other types of projects, the software engineering group began by making a distinction that provided context for the rest of their discussion. This group limited their discussion to projects from the computational science community. These projects were defined as being focused on conducting science or gaining new knowledge and typically written for large machines. Projects that were not addressed were those from the business community. These projects were defined as being focused on making money and increasing their customer base and typically written for smaller machines.

What are the top things that the software engineering community has to offer the HPC community?

The main contribution that the software engineers thought they could make to the HPC community was to find cases where Computational Science and other HPC projects had successfully used good software engineering practices, and communicate those successes to the broader community. The groups that have good software engineering practices (e.g. version control, regression testing, and inspections) have mostly learned them the hard way (i.e. they were passed down by previous team members). So, they only use good processes if they happen to have been on a project that used them in the past. There are a series of effective, elementary practices which require only a small amount of effort to implement. Beginning with some of these practices is safe way to begin interacting with HPC projects and also to remove a boundary to HPC use (i.e. people avoid HPC programming because of the perceived difficulty). Some examples of these practices are: version control, unit testing, and regression testing.

Another area in which software engineers can contribute is in the software architecture and design areas. Software engineers understand the need to design software to account for attributes like maintainability and portability in addition to functionality and performance. Making concepts like component-based software engineering accessible to the HPC community by providing libraries and compilers would be a great contribution. Finally, taking the knowledge of how to use middleware and applying to simplify access to grids would be helpful.

What are some problems or frustrations you have had in trying to work with the HPC community or the research domain?

One of the main frustrations that software engineering researchers have faced has been the different focus that the HPC developers have. In general, the software developed for HPC applications is treated more like a secondary tool, with the focus being on the scientific paper that can be published with the results. Therefore, the software is often thrown away and not valued as an asset like it might be in the IT sector. Furthermore, the two communities have different views of the real problems with software development. For example, software engineers focus a lot of time and energy on quality assurance mechanisms, while many of these are largely ignored in the HPC community.

A second set of frustrations concerns the cultural divide between software engineers and HPC developers. It is common for software engineers to face the complaint from HPC developers that they are "just imposing more process on us", rather than just "letting us write our algorithms." In addition, there is the view that because software engineers do not understand the real problems, the HPC developers do not see the benefit of listening to them. This problem is worsened because there are many HPC projects that succeed without using formal software engineering (because of the small size or the presence of smart people) so they do not see the need for formal software engineering. But, often these projects are not followed through to the maintenance phase where the lack of formality really becomes an issue. One of the reasons for this divide is that software engineers are not always given access to enough HPC projects to allow them to understand what works and what does not. This is especially true for those projects that are not successful. One important way that software engineers learn is through the analysis of failures.

What are some things that software engineers would like to offer to the HPC community, but we cannot yet?

Understanding the differences between research projects and more traditional software projects, an import goal for the software engineering researchers is to offer a valid software lifecycle for re-

search projects. One issue that must be overcome by the software engineering researchers is that they often lack a deep understanding of how to actually write the HPC software. The software engineering researchers also see the need to provide more evidence to show which methods and tools work in which situations and which do not. This evidence needs to be as quantitative as possible, because pure anecdotal evidence does not carry much weight in the HPC community. So, in addition to offering the buffet of methods, software engineers need to offer sound advice on when to choose each option. One proposal for such a lifecycle is that the domain experts develop a version of the code that is then optimized by a software engineer who is an expert in parallelization. The process will need to provide for the verification and validation of the optimized code to ensure that the science or engineering embodied in the code is not altered by the optimization.

Summary and Road Ahead

This workshop brought together experienced software engineering researchers and experience HPC Application researchers and developers to share their experiences and discuss common issues. The workshop produced two important outcomes that affect future research.

1. The members of the HPC community agreed that identifying and articulating the need for a software lifecycle and tool set specifically tailored to research projects made the workshop successful.
2. Researchers need to examine other types of software development that has similar characteristics to HPC development to determine which approaches can be borrowed and tailored for HPC development. Some types of software that should be examined include:
    a. Internal projects (i.e. those projects that are developed for in-house use rather than for commercialization) that often provide tool support for developers working on commercial software. Because these projects are created to support a particular user group, who often take part in their development, they have to deal with changing requirements as new needs are identified and there is a large overlap between the users and the developers of the software. These two characteristics suggest that techniques and methods that have been found to be useful on internal projects have the potential to be successful on HPC projects.
    b. High-risk software – These projects have a risk of failure that higher than for other types of software (e.g. software using a new development paradigm, software for a new domain, software with an uncertain market or software with an undefined customer base). This increased risk should lead these projects to choose lifecycle models and development approaches that help mitigate the risk. The HPC applications face similar types of risk because they are exploring unanswered scientific questions and may simply fail due to incorrect assumptions about nature. Any risk-mitigation techniques that have been found to be useful should be investigated for use on HPC projects.

This workshop was educational and useful for members of both communities represented, software engineering and high performance computing developers and researchers. The interesting discussion, captured in this report, highlighted both the similarities and differences of the software development approaches taken by the two groups. As a result of this workshop, the software engineering researchers have a better understanding of the problems faced by members of the HPC community and the members of the HPC community have a better understanding of the types of expertise and support that software engineering can provide them. These mutual understandings should set the stage for future collaborations between software engineering researchers and HPC researchers and developers.

Acknowledgements

Special thanks to Lorin Hochstein for taking detailed notes during the discussion and providing editorial feedback on early versions of this report.

References

[1] Alameh, R., Zazworka, N., and Hollingsworth, J.K., Beyond Performance Tools: Measuring and Modeling Productivity in HPC, in *SE-HPC 2007 (Held at ICSE 2007)*. 2007: Minneapolis, MN.

[2] Beck, K., *Extreme Programming Explained: Embrace Change*. 2000, Reading, MA: Addison-Wesley.

[3] Gorton, I., Chavarria-Miranda, D., Krishnan, M., and Nieplocha, J., A High Performance Event Service for HPC Applications, in *SE-HPC 2007 (Held at ICSE 2007)*. 2007: Minneapolis, MN.

[4] Hudak, D.E., Ludban, N., Gadepally, V., and Krishnamurthy, A., Developing and Computational Science IDE for HPC Systems, in *SE-HPC 2007 (Held at ICSE 2007)*. 2007: Minneapolis, MN.

[5] McCracken, M.O., Wolter, N., and Snavely, A., Beyond Performance Tools: Measuring and Modeling Productivity in HPC, in *SE-HPC 2007 (Held at ICSE 2007)*. 2007: Minneapolis, MN.

[6] Panas, T., Quinlan, D., and Vuduc, R., Tool Support for Inspecting the Code Quality of HPC Applications, in *SE-HPC 2007 (Held at ICSE 2007)*. 2007: Minneapolis, MS.

[7] Willenbring, J.M., Heroux, M.A., and Heaphy, R.T., The Trillinos Software Lifecycle Model, in *SE-HPC 2007 (Held at ICSE 2007)*. 2007: Minneapolis, MN.