Experience Paper: Software Engineering and Community Codes Track in ATPESC

Anshu Dubey
Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, IL 60439
Flash Center for Computational Science
University of Chicago
Chicago, IL 60637
Email:adubey@anl.gov

Katherine M. Riley
Argonne Leadership Computing Facility
Argonne National Laboratory
Lemont, IL 60439
Email:riley@alcf.anl.gov

***Abstract*—The Argonne Training Program in Extreme Scale Computing (ATPESC) was started by Argonne National Laboratory with the objective of expanding the ranks of better-prepared users of high-performance computing (HPC) machines. One of the unique aspects of the program was inclusion of a software engineering and community codes track. The inclusion was motivated by the observation that the projects with a good software process were better able to meet their scientific goals. In this paper we present our experience in running the software track. We discuss the motivation, reception, and evolution of the track over the years. We welcome discussion and input from the community to enhance the track in ATPESC and to facilitate inclusion of similar tracks in other HPC-oriented training programs.**

I. Introduction

In 2013 Argonne National Laboratory started a two-week summer training program called "Argonne Training Program on Extreme Scale Computing" (ATPESC) [1]. The program was originally funded for three years; it has been extended for another three. Motivation for a training program aimed at extreme computing came from the observation that few users had the necessary expertise to use the hig-performance computing (HPC) machines effectively. A significant fraction of users were largely ignorant of the challenges of using HPC machines and had difficulty meeting their science targets. These machines are expensive and rare resources; therefore, good utilization is critical, not just for the facilities, but also for advancing science. The reason for the existing wide gap between what is needed and what is known is partly due to the interdisciplinary nature of most of the work that is carried out on HPC platforms. The topics relevant to HPC are typically not in curriculums even at the graduate level in most universities. The vast majority of practitioners have learned what they need "on the job." ATPESC's aim is to reduce the gap by training young scientists, students, and other practitioners of scientific computing and to cultivate more informed and capable users in diverse scientific disciplines. The participants are enthusiastic about the program because it has had a significantly positive impact on their learning curve in using the machines.

The program's instruction format is a combination of lectures, keynotes, and hands-on exercises. The instructors include leading researchers and practitioners from various areas of HPC, such as MPI, OpenMP, math libraries, and data management. In this paper, we focus on a two-day track in the program that covers software engineering for HPC and community codes. Software engineering has a history of being underrated in the computational science community. As codes have grown more complex, the need for software engineering is being felt in more science domains that use computation. In discussing the track we have dual objectives of disseminating our insights and seeking input to further enrich the program.

We give a more detailed description of the motivation behind including the track in Section II, with its objectives described in Section III. Sections IV and V respectively describe the reception and feedback from the students and the evolution of the track. In section VI we give our insights and conclusions.

II. Motivation

The users of large-scale computing facilities come from many scientific research domains with wide variation in their software development and engineering practices. The software engineering practices range from none to robust. To a great degree the users' success in achieving scientific goals is directly related to the sophistication of their simulation planning and code readiness. Because of this correlation, HPC facilities often demand that the users demonstrate a minimum software maturity and planning ability before they can be allotted substantial compute resources. Programs exist to help new users learn necessary technologies and methodologies. These programs help, but they are not enough. The coverage is seldom comprehensive because of resource and expertise

constraints. ATPESC is compelling because it is comprehensive in its coverage of topics relevant to HPC and it casts its net wide for instructors. The lecturers and keynote speakers in the program are acknowledged experts. Many instructors go beyond the specifics of the tools to give insight to the students that can prepare them for self-learning if needed.

From the outset ATPESC placed appropriately large emphasis on the software engineering track. Software engineering practices that are critical for productivity include use of repositories, configuration and build process, testing and verification, and documentation. Practices such as provenance are important for reproducibility and credibility of science results. One topic that we include is rarely covered elsewhere: simulation planning. Being able to estimate resources needed to achieve scientific goals is a nontrivial activity. Equally important is to adequately plan for running the simulation campaign. This includes curating and archiving data, monitoring the state of simulation, and maintaining the equivalent of a laboratory notebook. All these topics are included in the track. Unlike some of the other tracks whose importance is well understood by the attendees, the software track has had the additional challenge of convincing a substantial fraction of skeptical attendees of its relevance early on. For this reason we turned the first session of the track into a motivating session, and we have retained this feature over the years. In the 2016 session we noticed for the first time that a majority of students were aware of many software engineering best practices and understood their usefulness in their scientific endeavors. Therefore, we may reduce the emphasis we have placed thus far on the motivating lectures. They may be replaced by more advanced topics or a more in-depth coverage of some of the topics.

The track also includes a section on community codes because they have been playing an increasingly important role in science through simulations. As scientific insight has grown over the years, the models being computed have grown more sophisticated and are often heterogeneous. Also, the scientific process is moving toward the integration of simulation and observations/experiments. As a result, both the codes and the workflows have grown more complex and are rarely within the reach of a single researcher or a small group. This handicap is reduced in those fields where substantial community resources exist. Recognizing this, more communities are moving toward pooling resources in terms of expertise and code components. We believe that current and future HPC users should be made aware of the advantages of resource-pooling and use of community codes (if they exist) in their research domain.

III. Objectives

The primary objective of the software track in the training program is to evangelize about the software process, and expose students to the prevalent practices and methodologies. Our objective is not to promote a specific practice or methodology. Instead, we provide exposure to a wide range of practices and methodologies and demonstrate their impact by including experience-based examples. Our aim is to equip the users with the necessary information that enables them to select suitable practices for themselves. In general our objectives can be summarized as follows:

- Emphasize the importance of software and scientific process for robust and credible science results
- Outline characteristics of sustainable software
- Give an exposure to processes required for developing scientific codes
- Show attendees the approach and effectiveness of code cooperation in a variety of domains
- Summarize practices that have been known to work in large-scale computational science
- Show how to integrate topics covered earlier in the program with examples of real applications
- Illustrate some of the sociological and technical challenges of those approaches.

The software engineering and community codes track runs for two days. The first few presentations aim to convert the skeptics – to make them understand that for credible science having a robust software and scientific process is a necessity, not a choice. These presentations lay out the objectives and motivations of the track for the students. The remainder of the lectures covered topics that can be broadly categorized under two headings: applications and process. The presenters in the applications category described their community, its goals, and its software and scientific processes. The process category then generalized individual topics in software engineering. Thus the applications presentations on the first day of the track motivated the detailed discussion of the process presentations on the second day. Below is a sample of applications and process presentations (from 2015).

- The impact of community codes in astrophysics
- Designing scalable scientific software (plasma)
- Modern features of production scientific code (Industrial combustion)
- NAMD (molecular dynamics)
- HEP complex workflows
- HACC: Application performance across diverse architectures (cosmology)
- Architecting community codes
- Software engineering practices
- Types of workflows
- Swift as a workflow solution
- Data provenance

Based on feedback from the past few years and the evolution in the status of software engineering in scientific computing, the track has undergone substantial changes that are reflected in the 2016 schedule (see Section V for specifics). Also, in 2014 and 2015 each, a half-day session was devoted to Software Carpentry-based material for basic software engineering practices. These presentations were greatly appreciated by the attendees. For the coming year this component of the track is being developed collaboratively between the IDEAS [2] project and various HPC facilities. The presentations prepared under this collaborations include the basic concepts covered

by Software Carpentry with customizations where needed for using HPC platforms for large scale scientific software. Details of this effort are described in Section V.

IV. Reception and Feedback

The reception of the track has varied from year to year and also within a year. This was expected because the attendees come from diverse backgrounds. The program is promoted in all possible venues of HPC and gets applications from all over the world, in all computational science areas and many science and engineering domains. Research communities themselves vary in the degree of penetration by software engineering practices and awareness of software sustainability issues. Communities such as astrophysics and computational chemistry have had a long history of community development, and students from these communities are well aware of the value of community codes and software engineering. On the other hand some communities are new to HPC (e.g. HEP experimentalists) and have little understanding of these concepts. Because the program is fairly comprehensive, the focus of interest for different students also varies. In addition to the track as a whole, the individual presentations have similar deviations. This is especially true of presentations about specific application areas or a specific tool. Below are a few highlights that illustrate the general trend of the comments.
"*I was impressed by the breadth of content in the community codes track*"
"*Quite relevant, I've been using a large community code so was able to appreciate the intricacies and challenges for such large codes*"
"*This session was quite interesting and had me think on this topic deeper*"
"*One of the more relevant tracks as I contribute to a community code*"
"*I think the data flow layout and data Provenance are particularly meaningful*"
"*Seriously, get rid of the talk on data provenance*"
"*Most of the talks are difficult to catch up*"
"*I think the track could be reduced to one day*"
"*This track was not as relevant as previous tracks*"

From these statements we can make a few interesting observations. One is that in the same year, two attendees can provide feedback completely opposite to each other. The comments about the data provenance presentation are an example, where one attendee found it to be among the most useful, while another attendee recommended eliminating it completely. Another observation is that the students are more likely to have a positive reception of the track if they have had some exposure to community projects. Because of such divergence in the feedback we include a healthy dose of our own judgment in incorporating suggestions from the students.

In general the track has received more positive comments than negative, with a small percentage of students finding the track to be not relevant or important. A few attendees mentioned that they had not thought about software engineering issues but that because of the track they will pay more attention to such issues. Others found it lacking in tangible learning moments. One common observation was that having several application-area talks did not add value. They seemed repetitive because their software practices and challenges were similar. This was one of strongest motivators behind changing the contents and the format of the track from 2016.

V. Evolution

When the program began, software engineering in HPC had gained limited traction. By and large the developer groups that deployed these practices were those that had learned from hard experience. Either they were a large and geographically diverse group, or their software had too many interacting but independently moving parts: in short, those groups for whom the management and the reliability of their software would have been intractable without instituting a process. As could be expected, several students of the time had little or no exposure to even rudimentary practices such as version control and automated testing. However, within the next couple of years, software sustainability and productivity began to appear frequently in computational science discussions because of confluence of several factors. One was growing influence of software carpentry [4]; another was the SI2 initiative by the NSF [3] and a set of workshops on this topic [5], [6]. Yet another was a growing concern about lost productivity due to fragmented software efforts and massive amounts of software replication. Some public retractions of publications in prestigious journals due to errors caused by a lack of adequate testing and verification also made it obvious that business as usual was no longer an option if computational science was to be an instrument of scientific discovery.

Over the years the baseline awareness of software engineering practices among the students of the program has risen. A few basic practices such as use of repositories and some form of regular testing have become more of a norm rather than an exception. The curriculum of the track has evolved to reflect this change. A major change from the previous years is in the number of application codes represented in the program. In the early years motivating the students was critical, as was giving them exposure to the practices adopted by the groups who had faced software engineering challenges and found some solutions that worked for them. As the landscape has changed, so has the students' understanding of the need for software engineering among the practitioners. Additionally, this was the component of the program that the students found to be repetitive and less useful. Therefore, instead of having several applications present their insights and practices, we changed the format to include lectures on best practices in software engineering in HPC followed by one application presentation that ties all the practices together. Examples from specific applications appear in the best practices lectures, but the applications themselves are not the focus.

The generation of best practices lectures was facilitated by the IDEAS project funded by the U.S. Department Of Energy

Office of Advanced Scientific Computing Research (DOE-ASCR). The numerical methods and math library track has been supported and run by the SciDAC FastMath institute funded by DOE-ASCR. In a similar spirit the software engineering track has become adopted by the IDEAS project since 2016. The IDEAS team has huge cumulative expertise and project experience. An engagement from a group of people who deal with many of the software productivity issues on a regular basis has exposed and filled many gaps in the earlier content. The current content has been systematically developed through close collaboration between the software experts from the IDEAS project and the training and operations personnel in the DOE computing facilities. The content developed through this collaboration was presented earlier in a series of webinars, and the response has been overwhelmingly positive. The overall registrations have exceeded 250, while the attendance of various webinars has fluctuated between 70 and 150. A great deal of substantive feedback from the attendees has been incorporated into the material. The covered material overlaps a great deal with the Software Carpentry material and because it is prepared by veteran HPC practitioners, it brings in insights about specific challenges in using such platforms. Because of the overlap and higher relevance of this content for HPC users, this material replaces the Software Carpentry component. To provide an idea of the extent of changes in the program we list below the lecture topics in 2016.

- Repositories and continuous integration
- IDE/configuration/building/deploying
- Testing and documentation
- Software refactoring
- Reproducibility
- Prevalent software engineering practices in modern codes
- Impact of community codes
- Sociology of community development
- Software and process design for future
- Planning Simulations
- Workflow/data curation and Provenance

The group of students in the 2016 session were not only aware if software engineering practices, but also had self-awareness to know that in many ways their process was not yet the best possible. Because of this there is likely to be substantial changes in several of the lectures in future. In 2016 they were largely introductory; in future they are more likely to include advanced topics. A couple of lectures such as impact of community codes and sociology of community codes may be combined into one. We mention these proposed changes to reiterate that the evolution of the track must reflect the rapidly evolving state of knowledge about software engineering issues in the computational science community to stay relevant and useful to the students.

VI. Conclusions and Discussion

The inclusion of the software engineering and community codes track in ATPESC has been rewarding to both the facilities and the attendees: the facilities, because they have to do less hand-holding for better prepared users and better managed software, and the attendees because they get farther in their work when they use well -managed software. The reception has been largely positive from the beginning, with the appreciation of the track growing steadily over the years. This growth is due to both the evolution of the track and the increasing awareness of a need for a good software process in the computational science community.

One of the reasons for presenting this experience report is to seek input for further enhancement of the track. We welcome suggestions about the organization and the content. We are also interested in new ideas about the topics we cover and new topics that might be interesting and useful that we do not yet include. We also hope that sharing our experiences will initiate further discussions that will help enhance our program and similar training programs. The positive reception of the track indicates that software engineering for computational science should be an integral part of any curriculum targeted at scientific software.

Acknowledgments

This work was supported by the U.S. Department of Energy Office of Science Office of Advanced Scientific Computing Research.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (Argonne). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.

References

[1] Argonne training program on extreme scale computing. https://extremecomputingtraining.anl.gov/.
[2] IDEAS productivity - howto documents. https://ideas-productivity.org/resources/howtos/.
[3] Implementation of NSF CIF21 software vision (sw-vision). http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504817.
[4] Software Carpentry. http://software-carpentry.org/.
[5] Software productivity for extreme-scale science. https://www.orau.gov/swproductivity2014/SoftwareProductivityWorkshopReport2014.pdf.
[6] A. Dubey, D. Q. Lamb, and E. Balaras. Building community codes for effective scientific research on HPC platforms. http://flash.uchicago.edu/cc2012, 2012.