Teaching Software Sustainability for High Performance Computing at ATPESC

Anshu Dubey*†, Katherine M. Riley‡, and David E. Bernholdt§
*Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, Email:adubey@anl.gov
†Department of Computer Science, University of Chicago, Chicago, IL 60637
‡Argonne Leadership Computing Facility, Argonne National Laboratory, Lemont, IL 60439, Email:riley@alcf.anl.gov
§Computer Science and Mathematics Division and National Center for Computational Sciences,
Oak Ridge National Laboratory, PO Box 2008, MS 6164, Oak Ridge, TN 37831-6164, Email: bernholdtde@ornl.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 track on software engineering and community codes. The inclusion was motivated by the observation that the projects with good software processes were better able to meet their scientific goals. Over the years, with greater awareness of software sustainability issues in the community, the track has evolved into a software productivity and sustainability track. In this paper we present our experience in choosing and disseminating the content related to the topic of software engineering in high performance computing science from the beginning of the program until now. We discuss the motivations and the reception of the tracks. We also document the evolution of the track over the years based on student feedback and also the growth of awareness about software productivity in high performance computing.**

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 since been extended twice, with the latest extension under the aegis of the U.S. Department of Energy's Exascale Computing Project (ECP). The original motivation for a training program aimed at extreme computing came from the observation that few users had the necessary expertise to effectively use the leadership class machines hosted at Argonne. A significant fraction of users were largely ignorant of the challenges of using high performance computing (HPC) machines, and had difficulty meeting their science targets. Such 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 wide existing 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 curricula even at the graduate level in most universities. The vast majority of practitioners learn 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. 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 in future, as needed. The participants have been enthusiastic about the program because it has 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 disciplines relevant to HPC such as programming models, math libraries, data management and visualization, use of accelerators, machine learning, workflows and so on. In this paper, we focus on a track that started out as a two day track covering software engineering for HPC and community codes, and has since been converted into a single day track covering software productivity and sustainability. This change occurred when the ECP project started sponsoring the program. The time-frame was reduced for two reasons. The first is that several new topics have acquired importance in HPC that they did not have in the beginning. These include an increased focus on accelerators, and machine learning and artificial intelligence. The second reason is that several introductory lectures that were necessary in the early years are no longer needed because of greater awareness of software sustainability issues in the HPC science community. 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 evolution of the track and the reception and feedback from the students. In section VI we present our insights and conclusions.

II. Motivation

The users of large-scale computing facilities come from many scientific research domains with large variation in their software capabilities, and its development and engineering practices. It is relatively rare that users at this level are merely using software developed by others. Most are, at some

level, also developers of the codes they run. Likewise, while there is some relatively simple and easy to maintain and use software used at such facilities, the dominant form is highly complex and multidisciplinary. This is because leadership class facilities typically address the more complex questions raised in scientific research, and complex problems requires more varied models and approaches for their solution. Among the complex codes, the software engineering practices range from none to very 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. However the primary focus of these demands, and the facilities' training efforts to help users meet the bar has been around concerns of performance, scalability, and the ability to effectively utilize the center's computational resources. These training programs help, but are not enough. The coverage is seldom comprehensive because of resource and expertise constraints. And in particular, software engineering practices are rarely included.

A. Software Engineering

From the outset ATPESC placed an 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, documentation, and licensing. Practices such as provenance are important for reproducibility and credibility of science results. One topic that we have included in some years is rarely covered elsewhere; simulation planning. Being able to estimate needed resources to achieve scientific goals is a non-trivial activity. It is equally important to adequately plan for running the simulation campaign. This includes curating and archiving data, monitoring the state of simulation, and maintaining an equivalent of laboratory book. All the topics mentioned above have been covered in the track.

Unlike some of the other tracks in ATPESC, 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 focused the first session of the track on motivating the topic, and we retained this feature for several years. In 2016 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. This has led to reducing the emphasis we had previously placed on the motivating lectures. Also, the class of 2016 turned out to be far more advanced than all the previous years. The feedback from this class has lead us to reconsider the topics covered in the track. As a result the level of lectures has been raised to mid- to advanced- level while minimizing the introductory level material. The track has been redesigned, and has started leveraging the content developed by the IDEAS project, funded initially by the Office of Advanced Scientific Computing Research (ASCR) in the U.S. Department Of Energy (DOE) and later by the ECP [4].

B. Community Codes

In the first few years the track also included a section on community codes because they had been emerging in importance 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 towards 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 there are substantial community resources.

An example is the astrophysics community where the culture of publicly available codes has existed for many years [16], [11]. This has been tremendously helpful for the not just the rate of scientific discovery but also for scientific sophistication reached by students during their graduate studies. Because they do not have to write every component of the code from scratch for themselves, the students can set higher scientific goals for themselves from their simulations. Moreover, the codes they use are more robust and better tested than their own codes would have been because many people would have used the same code and found and reported more defects than any individual can.

Recognizing this, more communities are moving toward pooling resources in terms of expertise and code components. We believe it is important for current and future HPC users to be aware of the advantages of resource-pooling and use of community codes if they exist in their research domain. It is particularly important to bring this awareness to young scientists, because they are more prone to developing their own code from scratch. They must be made aware of the cost of reinventing the wheel, not only in terms of time spent, but also in terms of the sophistication of the tools being used in their research. In the early years of the program, when there was a greater need to bring this lesson home to students, there were at least half a day of lectures representing various community codes. More recently, we have found the value of community codes to be fairly well understood within the community, including the ATPESC audience, so the substantial focus on this topic has been reduced and, eventually eliminated over time.

III. Objectives

The primary objective of the software engineering/software productivity and sustainability track in the ATPESC program has been to evangelize a thoughtful software development process. Our objective has never been to promote a specific practice or methodology. Instead, we have focused in providing exposure to a wide range of practices and methodologies, and demonstrating their impact by including experience-based examples. Our aim is to equip the students with the necessary information to enable them to select appropriate practices

for their own work and projects. The objectives themselves have evolved with the changing community perceptions of software productivity and sustainability. At the beginning of the program our objectives could be summarized as follows:

- Emphasize the importance of software and scientific process for robust and credible science results.
- Show attendees the approach and effectiveness of code cooperation in a variety of domains.
- Give an exposure to processes required for developing scientific codes
- Summarize practices that have been known to work in large-scale computational science.
- Illustrate some of the sociological and technical challenges of those approaches.

In the beginning, when the software engineering and community codes track was a completely new concept in HPC training, the first few presentations were aimed at the skeptics. They were made by users and developers of community codes that had adopted prevalent best practices out of necessity and found them to be beneficial to all aspects of their work, including better science. The idea was to make the skeptics understand that for credible science having a robust software and scientific process is a necessity, not a choice. The presentations laid out the objectives and motivations of the track for the students. The remainder of the lectures until 2015 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.

In the third year of the program we began to notice a change in the student population. Students were a lot more familiar with the ideas of good software practices and the material we had been using until then had become too low level for a majority of students. This awareness coincided with the start of the IDEAS project with focus on software productivity for high performance scientific application. The IDEAS project was tasked with developing methodologies for incorporating software engineering practices in participating projects. A great deal of the work done in the IDEAS project and the objectives of the project were relevant to the ATPESC software track. Therefore, to address the change in the student body, and because of the synergy with the project, the objectives of the track also became informed by those of the IDEAS project. Not that the any of the existing objectives changed, instead, their prioritization changed and a few more were added. The awareness of software sustainability as being critical for scientific achievement in HPC is a relatively new phenomenon, and with rapid evolution of HPC itself, the concepts and understanding of software sustainability are also evolving. Therefore, our objectives at any time are at best a snapshot of what this field means to the HPC community. At this writing the objectives can be summarized as follows.

- Outline characteristics of sustainable software, and provide information about how can it be achieved.
- Emphasize the importance of software and scientific process for robust and credible science results
- Make attendees aware of the importance of reproducible science in HPC.
- Expose attendees to processes required for developing reliable scientific codes
- 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.

IV. Evolution

When the program began, software engineering in HPC had very limited traction. Many groups applying for computing time through major allocation programs such as the U.S. Department of Energy's (DOE) Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program [3] were barely prepared to utilize their allocations, and some were even unable to meet their scientific goals because of their lack of preparation. There was limited awareness of the role played by a good software process in achieving scientific objectives in HPC. By and large the developer groups that deployed software processes were those that had learned from hard experience. They were either a large and geographically diverse group, or their software had 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 from the students of these times, many of them 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 [5]; another was the U.S. National Science Foundation's Software Infrastructure for Sustained Innovation (SI2) initiative [2] and a set of workshops on this topic [6], [14]. 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 earliest years was at first in the number of application codes represented in the program. In the earliest years motivating the students

was critical, as was giving them exposure to the practices adopted by the groups that had faced software engineering challenges and found some solutions that worked for them. As the landscape changed, so did the degree of penetration of understanding the need for software engineering among the practitioners. Additionally, this was the component of the program that the students found to be the most repetitive, and therefore 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 tied all the practices together. The examples from specific applications appeared in best practices lectures, but the applications themselves were not the focus.

The generation of best practices lectures was facilitated by the IDEAS project. The IDEAS team had huge cumulative expertise and project experience. An engagement from a group of people that deal with many of the software productivity issues on a regular basis exposed and filled many gaps in the earlier content. The second generation of the content was systematically developed through close collaboration between the software experts from the IDEAS project and the training and operations personnel in the DOE computing facilities. This material was first presented in a series of webinars, and the response was overwhelmingly positive. The "Best Practices for HPC Software Developers" series [9] has now had more than 40 webinars, with more than 3000 unique registrants and an average of more than 80 attendees at each webinar. The success of the series lead to the adoption of several of the webinars in ATPESC software track. Below we list the topics covered both in the webinars and the ATPESC software track.

- 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

To our surprise this same material from the webinars proved to be at too low a level for the vast majority of students in the track. It became apparent even as the lectures were being delivered that introductory material was redundant for most of the students. While it is certainly possible that this group of students were exceptional and their level of knowledge was just a blip, we believe that there is a genuine upward trend. Years of efforts by devoted advocates in academia and funding agencies seem to be finally making an impact. Open source development is becoming ubiquitous and a mechanism by which productivity is judged. As scientific results have come under greater scrutiny it has become important to be able to reproduce results. Initiatives by major conferences and journals have also brought focus on software process. We finally felt that we could move away from justifying the track to the students, instead, we could focus on the actual training content at mid and advanced levels.

The material for 2017 track was drawn from the topics covered in an invited tutorial on software productivity at the SIAM Computational Science and Engineering (CSE) conference. The track was reduced to one day with the assumption that students have knowledge of base software engineering practices obviating the need to spend time on introductory level material. Topics covered in this iteration of the track are listed below.

- Overview and objectives
- Git introduction with hands on exercises
- Agile methodologies via Kanban and GitHub
- Reproducibility
- Testing, verification and continuous integration
- Software lifecycle with an example, community impact
- Licensing

From 2017-2019 incremental changes occurred in these presentations as some of the material started to be presented regularly in tutorials by the IDEAS project at venues such as Supercomputing (SC) and International Supercomputing Conference (ISC) conferences and the ECP project's Annual Meeting (with approximately 800 attendees). The changes were mostly in the form of updated material as the software sustainability field itself evolved, and inclusion of real world examples of applying the methodologies and principles taught in the material. These examples are derived from the software projects experience of various members of the IDEAS team. For example, in the design module the collective experience of framework design in several community codes [13], [18], [12], [20], [10], [7] is captured in a module. Similarly in the testing module an example has been added for generating tests in legacy codes [17], and for building a granular test suite that also provides coverage for interoperability [8]. A detailed example for code refactoring is based on the work reported in [19], [15].

One aspect of the both the ATPESC track and all the tutorials has consistently been called out in student feedback, and that is the very limited hands-on component. To address this criticism in the 2020 ATPESC edition we leveraged a running example of solving heat equation from another track in the program. In the Numerical Methods and Math Libraries track this example is used to highlight various methods that can be used to solve the heat equation through the use of different libraries. Since this track precedes our track, we reasoned that this would be a good example to use for our hands-on component as well. We used this example to explain agile methodologies for project management, code architecture design, checking for code coverage in testing and continuous integration, and code refactoring. We created a monolithic version of the solution in a single file and built simple exercises for students that would result in modular extensible code with adequate testing coverage.

V. Reception and Feedback

The reception of the track has varied from year to year, and of course spans a spectrum in any given year. This is 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 computational science areas covering many science and engineering domains. Research communities themselves vary in acceptance of 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 there are domains that are relatively new to computations and have very little appreciation for the value of good software. Because the overall training program is fairly comprehensive, the focus of interest for different students also varies. In addition to the track as a whole, the individual presentations also have similar spread. This was especially true in the early years in presentations about specific application areas or a specific tool. Interestingly, in the same year, different attendees would typically provide feedback completely opposite to one another for the same presentation. For example in year 2, one attendee found data provenance presentation it to be among the most useful, while another attendee recommended eliminating it. A general observation was that the students were more likely to have a positive reception of the track if they had had some prior exposure to community projects. Students coming from narrowly defined fields tended to view the track as not particularly useful. Because of such variation in the feedback we included a healthy dose of our own judgement in incorporating suggestions from the students.

Even with a not insignificant fraction of students viewing the track with skepticism, the track was viewed as serving an important purpose. For many attendees this was the first time they became aware of the challenges of sustainability and reproducibility, both of which have become increasingly important to scientific computing. Many students mention that they had not thought about software engineering issues, but the track motivated them to pay more attention. Since 2017, with the new material, the track has had a higher level of engagement from the students, and the feedback has been largely positive. Every year further improvements have been made based on attendee feeback from all venues where we present this material. This year, the track has had overall rating comparable to tracks on programming models. It is among the most highly rated tracks.

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 students 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 grew steadily over the first few years. This growth was due to both the evolution of the track and the increasing awareness of a need for software process in the computational science community. The emphasis and the content of the track has to continually evolve to maintain its relevance and to sustain interest from students.

Acknowledgments

We wish to acknowledge the extensive contributions of the people who have developed material and presented in the ATPESC software track over the years, as well as the IDEAS tutorial team.

This work was supported in part by the U.S. Department of Energy Office of Science Office of Advanced Scientific Computing Research. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

ORNL is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] Argonne training program on extreme scale computing. https://extremecomputingtraining.anl.gov/.

[2] Implementation of NSF CIF21 software vision (sw-vision). http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504817.

[3] Incite leadership computing. https://www.doeleadershipcomputing.org/.

[4] Interoperable design of extreme-scale application software (IDEAS). https://ideas-productivity.org/.

[5] Software Carpentry. http://software-carpentry.org/.

[6] Software productivity for extreme-scale science. https://www.orau.gov/swproductivity2014/SoftwareProductivityWorkshopReport2014.pdf.

[7] A. Dubey and D. Graves. A design proposal for a next generation scientific software framework. HeteroPar'2015, colocated with Europar-2015.

[8] A. Dubey, K. Weide, D. Lee, J. Bachan, C. Daley, S. Olofin, N. Taylor, P.M. Rich, and L.B. Reid. Ongoing verification of a multiphysics community code: FLASH. *Software: Practice and Experience*, 45(2), 2015.

[9] D. E. Bernholdt, O. A. Marques, E. M. Raybourn, A. D. Barker, and R. J. Hartman-Baker. The HPC best practices webinar series. *Journal of Computational Science Education*, 10(1):108–110, January 2019.

[10] CASC. SAMRAI structured adaptive mesh refinement application infrastructure. https://computation.llnl.gov/casc/SAMRAI/, December 2007. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory.

[11] A. Dubey. A study of hydrodynamics based community codes in astrophysics. In G. Juckeland and S. Chandrasekaran, editors, *Tools and Techniques for High Performance Computing*, pages 89–97, Cham, 2020. Springer International Publishing.

[12] A. Dubey, A. Almgren, J. Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Löffler, B. O'Shea, E. Schnetter, B. V. Straalen, and K. Weide. A survey of high level frameworks in block-structured adaptive mesh refinement packages. *Journal of Parallel and Distributed Computing*, 74(12):3217–3227, 2014.

[13] A. Dubey, K. Antypas, M. Ganapathy, L. Reid, K. Riley, D. Sheeler, A. Siegel, and K. Weide. Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code. *Parallel Computing*, 35(10-11):512–522, 2009.

[14] A. Dubey, D. Q. Lamb, and E. Balaras. Building community codes for effective scientific research on HPC platforms. http://flash.uchicago.edu/cc2012, 2012.

[15] A. Dubey, J. O'Neal, K. Weide, and S. Chawdhary. Distillation of best practices from refactoring flash for exascale. *SN Computer Science*, 1(4):223, Jul 2020.

[16] A. Dubey, M. Turk, and B. O'shea. The impact of community software in astrophysics. In *Proceedings of WCCM-XI;ECCM-V;ECFD-VI, E. Onate, J. Olivier and A. Huerta editors*, 2014.

[17] A. Dubey and H. Wan. Methodology for building granular testing in multicomponent scientific software. In *SE4Science, held in conjuction with ICSE 2018, Gothenburg, Sweden*, 2018.

[18] J. Moulton, M. JC, and M. e. a. Day. High-level design of Amanzi, the multi-process high performance computing simulator. Technical report, DOE-EM, Washington, DC, 2012.

[19] J. O'Neal, K. Weide, and A. Dubey. Experience reort: Refactoring the mesh interface in flash, a multiphysics software. In *WSSSPE6.1, colocated with eScience 2018, Amsterdam, Netherlands*, 2018.

[20] E. Schnetter. Performance and optimization abstractions for large scale heterogeneous systems in the cactus/chemora framework, 2013.