

# Why effective software practices are essential for CSE projects

Presented at  
**Better Scientific Software tutorial**

**ECP 2<sup>nd</sup> Annual Meeting, Knoxville, Tennessee**

**Anshu Dubey**

**Argonne National Laboratory**



EXASCALE COMPUTING PROJECT

# License, citation and acknowledgements



## License and Citation

- This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/) (CC BY 4.0).
- Requested citation: Anshu Dubey , Why effective software practices are essential for CSE projects, tutorial, in Exascale Computing Project 2<sup>nd</sup> Annual Meeting, Knoxville, Tennessee, 2018. DOI: *TBA*.

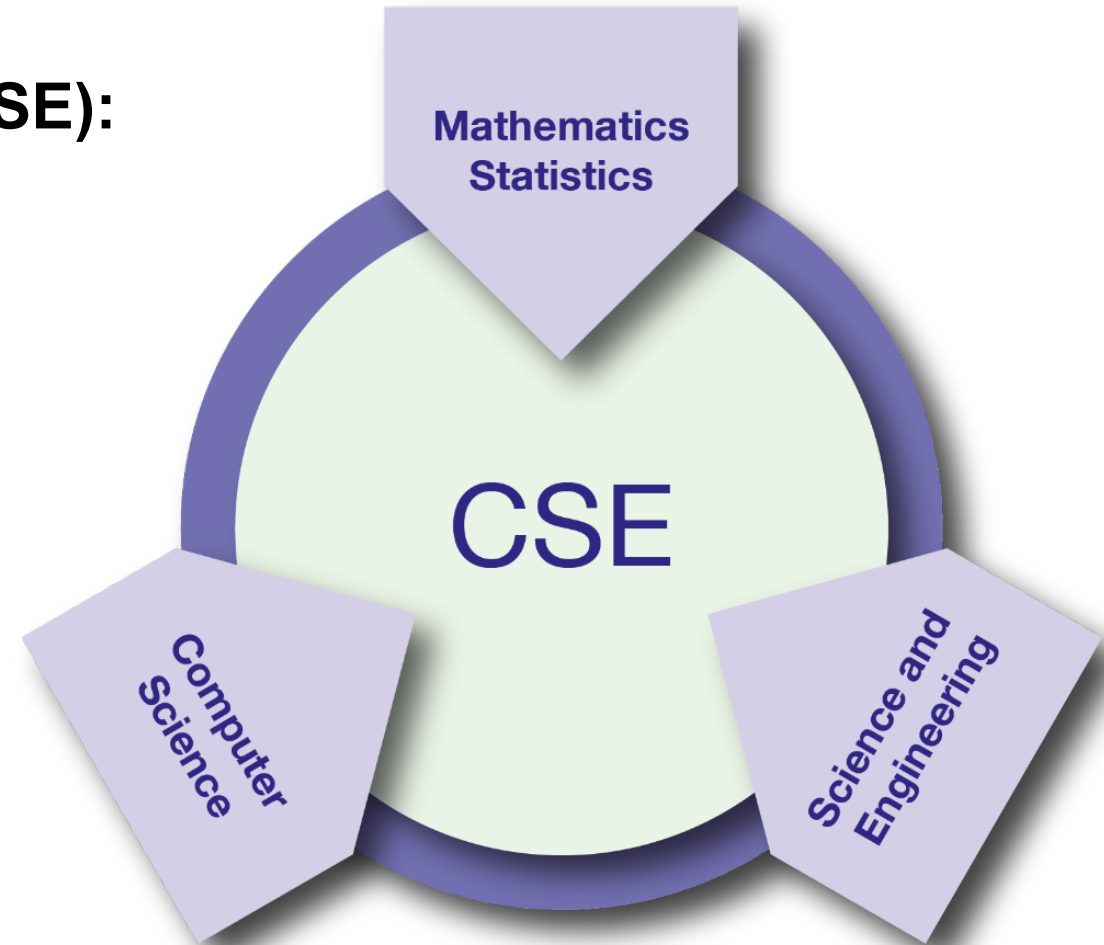
## Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and 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.

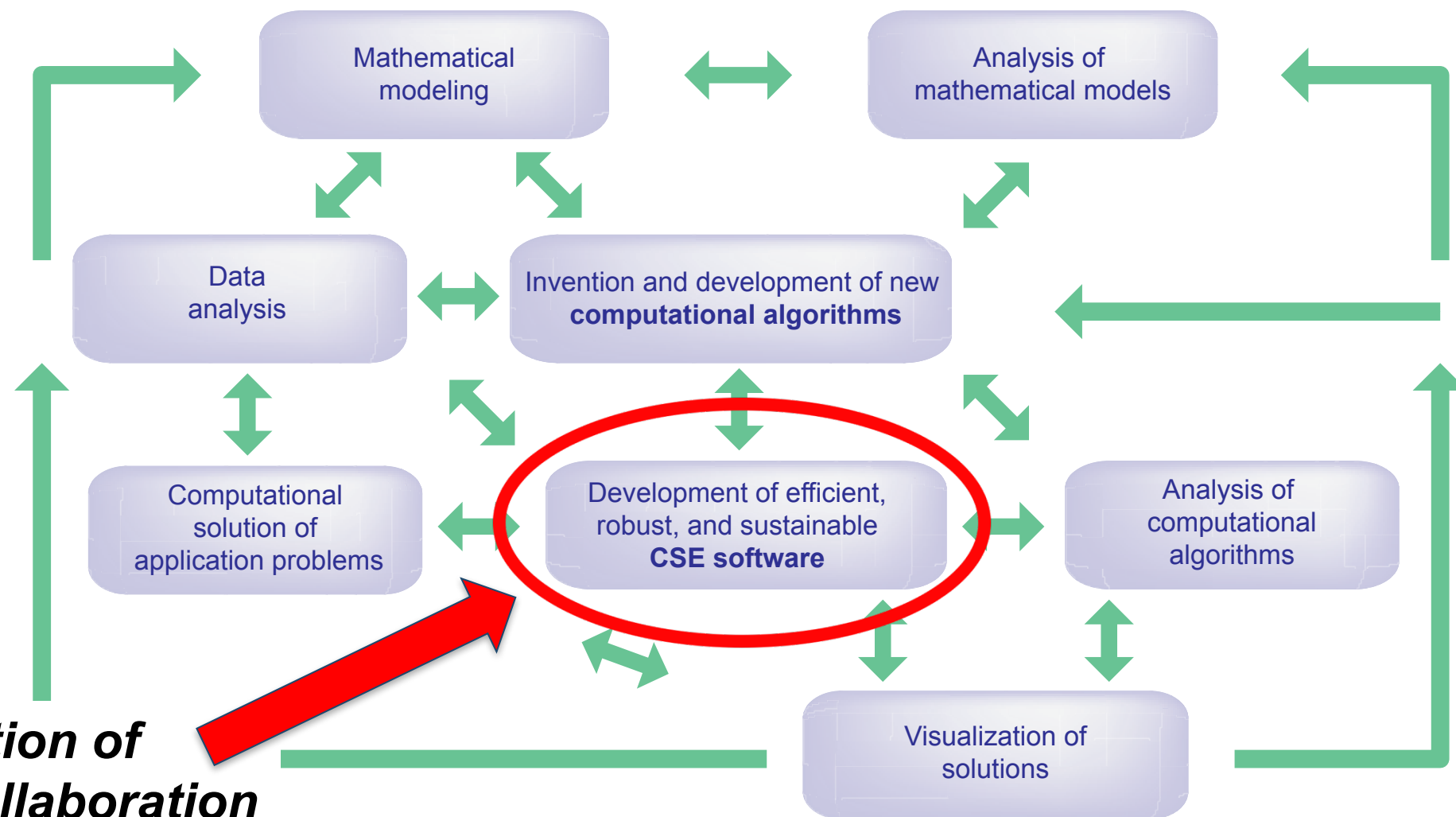
# What is CSE?

- **Computational Science & Engineering (CSE): development and use of computational methods for scientific discovery**
  - all branches of the sciences
  - engineering and technology
  - support of decision-making across a spectrum of societally important apps
- **CSE: essential driver of scientific and technological progress** in conjunction with theory and experiment

**Reference: Research and Education in Computational Science and Engineering**, U. Rüde, K. Willcox, L.C. McInnes, H. De Sterck, et al., Oct 2016, <https://arxiv.org/abs/1610.02608>



# Software is at the core of CSE



***Software: foundation of sustained CSE collaboration and scientific progress***

# Heroic Programming

Usually a pejorative term, is used to describe the expenditure of huge amounts of (coding) effort by talented people to overcome shortcomings in process, project management, scheduling, architecture or any other shortfalls in the execution of a software development project in order to complete it. Heroic Programming is often the only course of action left when poor planning, insufficient funds, and impractical schedules leave a project stranded and unlikely to complete successfully.

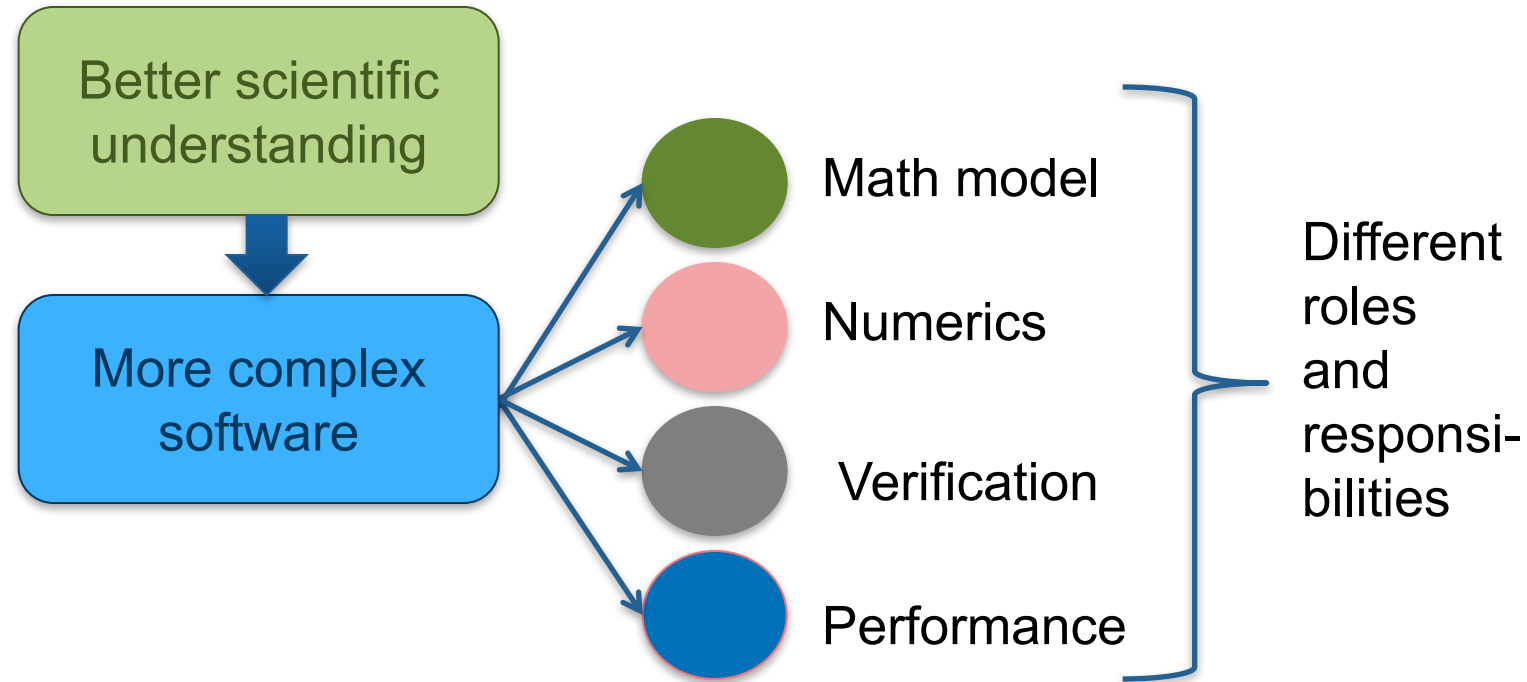
From <http://c2.com/cgi/wiki?HeroicProgramming>

**Science teams often resemble heroic programming**

Many do not see anything wrong with that approach

# What is wrong with heroic programming

Scientific results that could be obtained with heroic programming have run their course, because:



It is not possible for a single person to take on all these roles

# Other reasons

Accretion leads to unmanageable software

- Increases cost of maintenance
- Parts of software may become unusable over time
- Inadequately verified software produces questionable results
- Increases ramp-on time for new developers
- Reduces software and science productivity due to technical debt

consequence of choices – quick and dirty incurs **technical debt**, collects interest which means more effort required to add features.

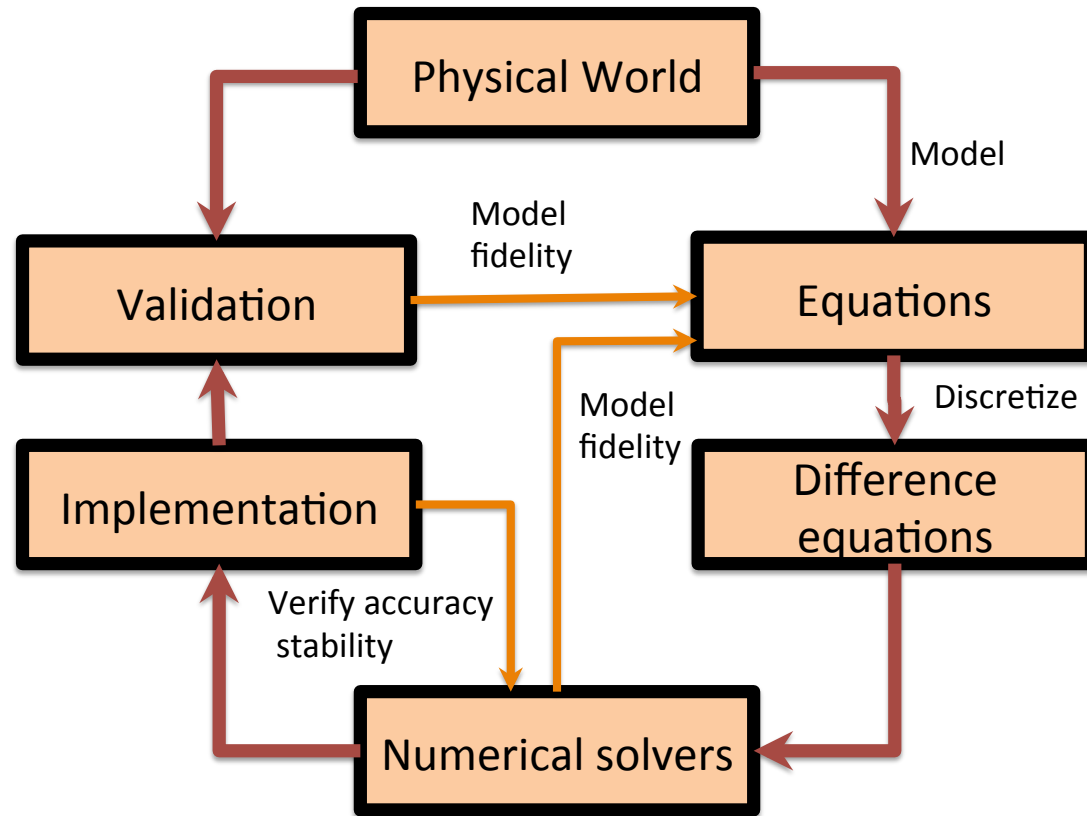
# Increasing complexity of CSE software

- Multiphysics and multiscale modeling
- Coupling of data analytics
- Disruptive changes in computer hardware
  - Requires algorithm/code refactoring
- Importance of reproducibility
  - Science requirements are unfolding, evolving, not fully known a priori

**Science through computing is only as good as the software that produces it.**



# Lifecycle



- Modeling
  - Approximations
  - Discretizations
  - Numerics
    - Convergence
    - Stability
- Implementation
  - Verification
    - Expected behavior
  - Validation
    - Experiment/observation

# Challenges of CSE software

## Technical

- All parts of the cycle can be under research
- Requirements change throughout the lifecycle as knowledge grows
- Verification complicated by floating point representation
- Real world is messy, so is the software

## Sociological

- Competing priorities and incentives
- Limited resources
- Perception of near-term overhead with deferred benefit
- Need for interdisciplinary interactions

# Taking stock: Understanding what you want from your CSE software and how to achieve it

- **Software architecture and process design**
  - Managing complexity and avoiding technical debt (future saving)
  - Worthwhile to understand trade-offs
- **Issues to consider**
  - **The target of the software**
    - Proof-of-concept
    - Discard once you're done with it (or the student/postdoc leaves)
    - Long-term research tool that successive group members will extend
    - Others ...
  - **How important are performance, scalability, portability** to you?
  - **Buy vs. build:** can you achieve your goals by contributing to existing software, or do you need to start from scratch?
  - What **3rd-party software** are you willing to depend on?
- **Target should dictate the rigor of the design and development process**
  - Considering resource constraints

# Software process for CSE

## Baseline

- **Invest in extensible code design**
  - Most uses need additions and/or customizations
  - Use version control and automated testing
  - Institute a rigorous verification and validation regime
  - Define coding and testing standards
- **Clear and well defined policies for**
  - Auditing and maintenance
  - Distribution and contribution
  - Documentation

## Desirable

- Provenance and reproducibility
- Lifecycle management
- Open development and frequent releases

# Customize according to *your* needs

- There is no “all or nothing”
- Focus on improving productivity and sustainability rather than purity of process
- Danger of being too dismissive too soon
  - Examine options with as little bias as possible
- Fine balance between getting a buy-in from the team and imposing process on them
- First reaction usually is resistance to change and suspicion of new processes
- Many skeptics get converted when they see the benefit

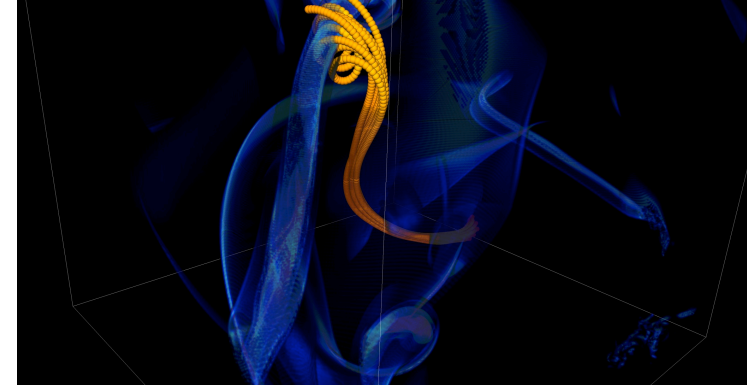
# Interdisciplinary Interactions

## A partnership model that works

- Science users treat the code as a research instrument that needs its own research
- Developers and computer scientists interested in a product and the science being done with the code
  - Helps to have people with multidisciplinary training
- Comparable resources and autonomy for the developers
  - And recognition of their intellectual contribution to scientific discovery
- Careful balance between long term and short term objectives

# What can happen without a process

- ❑ In 2005 BG/L was made available at short notice
- ❑ Quick and dirty development of particles
- ❑ Many in-flight corrections of defects
- ❑ One was adding tags to track individual particles
  - ❑ **Got many duplicated tags due to round-off**
- ❑ Had to develop post-processing tools to correctly identify trajectories



FLASH had a software process in place. It was tested regularly. This was one instance when the full process could not be applied because of time constraints. We got ready for the run in less than a month, the run went for 1.5 weeks, and it took over 6 months before we could trust the processed results.

# Resources

**Key:**

Blue text: covered in this tutorial

Black text: pointers to other resources



**Better Planning:**

- Requirements
- Design
- Software interoperability

**Better Development:**

- Documentation
- Version control
- Configuration and builds
- Deployment
- Issue tracking
- Refactoring
- Software engineering
- Development tools

**Better Performance:**

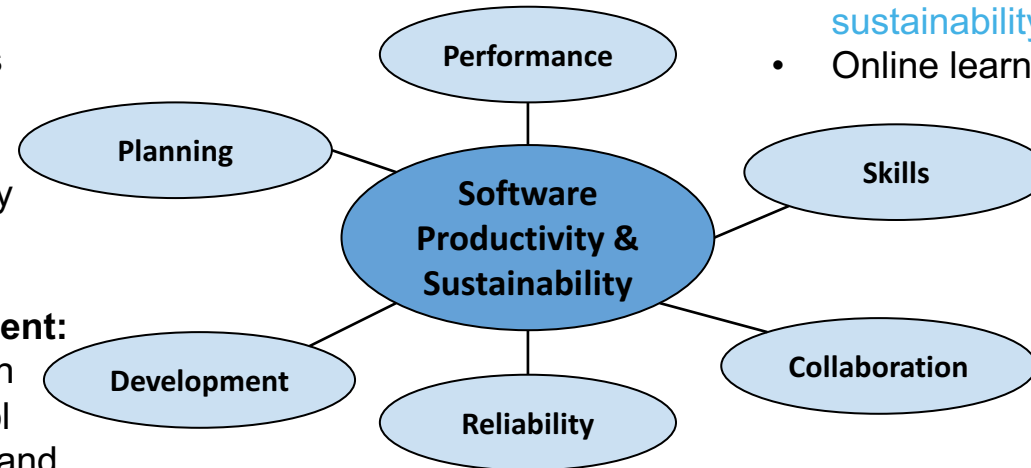
- High-performance computing
- Performance at LCFs
- Performance portability

**Better Skills:**

- [Personal productivity and sustainability](#)
- Online learning

**Better Collaboration:**

- [Licensing](#)
- [Strategies for more effective teams](#)
- Funding sources and programs
- Projects and organizations
- Software publishing and citation
- Discussion forums, Q&A sites



**Better Reliability:**

- [Testing](#)
- [Continuous integration testing](#)
- [Reproducibility](#)
- Debugging



# IDEAS *WhatIs* and *HowTo* documents

- **Motivation:** Software teams have a wide range of levels of maturity in SW engineering practices.

- **Resources:**

- ‘**What Is**’ docs: 2-page characterizations of important topics for CSE software projects
- ‘**How To**’ docs: brief sketch of best practices
  - Emphasis on “bite-sized” topics enables CSE software teams to consider improvements at a small but impactful scale
- Current topics:
  - *What Is CSE Software Productivity?*
  - *What Is Software Configuration?*
  - *How to Configure Software*
  - *What Is Performance Portability?*
  - *How to Enable Performance Portability*
  - *What Is CSE Software Testing?*
  - *What Are Software Testing Practices?*
  - *How to Add and Improve Testing in a CSE Software Project*
- More topics under development
- See: <https://ideas-productivity.org/resources/howtos>

**What Is Performance Portability for CSE Applications?**  
The IDEAS Scientific Software Productivity Project  
[ideas-productivity.org/resources/howtos/](https://ideas-productivity.org/resources/howtos/)

**Portability:** An application code is portable if it can run on a diverse set of platforms without needing significant modifications to the source and can produce predictably similar output.

**Performance portability:** An application code that exhibits similar performance across a wide range of diverse platforms at any given time. A code may need to run on a cluster with or without accelerators, or it may need to work on all the latest leadership computing platforms, each of which has a unique architecture and software stack. Therefore a baseline performance across a range of platforms is a fundamental requirement for these codes. When combined with the necessity of using scarce HPC resources well from the systems perspective, and time to solution and therefore scientific discovery from the scientific perspective, performance portability becomes a critical issue, especially in medium to large code bases.

**Software design approach:** A code designed for a detailed specific architecture is unlikely to be portable or performance-portable. A good practice has been to design for an abstract machine model with distributed memory and relatively shallow memory hierarchy. Solvers focused on maintaining spatial and temporal locality of data as much as possible without hard-coding any machine-specific parameters. Designing for abstract machine models is still a good practice, although more than one type may be needed. An option is to broadly characterize the target machines into a few abstract models as feasible, and even from those extract the commonalities for design considerations. For example, data can be organized so that compilers can vectorize, or hierarchical parallelism can be used to exploit coherence domains. C++ template programming provides one way of using abstractions.

**Focus on performance objectives:** A software project should have a clear outline of the performance objectives of the code that are important for scientific discovery. Performance considerations should be at the full application level, facilitated by tuning knobs. In general the tuning space of applications is large. Exposing tuning knobs and making them easy to set allow exploration of the performance space more quickly. For example, if accuracy requirements are known, then one can trade off accuracy within certain bounds for faster time to solution.

**Separation of concerns:** Designing software such that different expertise can concentrate on different aspects of the software is a good practice for many reasons; performance portability is among the most important. For example, isolating parallelism from the performance considerations of local sequential kernels has been useful. Similar encapsulation of functionalities so that different kinds of optimizations may apply to different sections of the code helps with portable performance.

**Composability:** A composable code is one that can select and combine existing components in the code base in many different ways to generate different applications. Composability also allows for multiple alternative implementations of selected code capabilities. This feature can be exploited to limit the amount of platform-specific implementation that needs to exist in a code.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.  
Version 0.2, April 25, 2016

**Impact:** Provide baseline nomenclature and foundation for next steps in software productivity and software engineering for CSE teams.

# Other Tutorials: Slides and video

## *Best Practices for HPC Software Developers*

- On-going monthly webinar series
  - <https://ideas-productivity.org/events/hpc-best-practices-webinars/>
  - Topics to date:
    - *What All Codes Should Do: Overview of Best Practices in HPC Software Development*
    - *Developing, Configuring, Building, and Deploying HPC Software*
    - *Distributed Version Control and Continuous Integration Testing*
    - *Testing and Documenting your Code*
    - *How the HPC Environment is Different from the Desktop (and Why)*
    - *Best Practices for I/O on HPC Systems*
    - *Basic Performance Analysis and Optimization*
    - *Python in HPC*
    - *Intermediate Git*
    - *Using the Roofline Model and Intel Advisor*

## *Argonne Training Program on Extreme-Scale Computing*

- Annual two-week short course
  - <https://extremecomputingtraining.anl.gov/>
  - **Software Engineering and Community Codes** track (2016) – 6 presentations
  - **Software Productivity** track (2017)
    - *What All Codes Should Do: Overview of Best Practices in HPC Software Development*
    - *Git Introduction*
    - *Better (Small) Scientific Software Teams*
    - *Improving Reproducibility through Better Software Practices*
    - *Testing and Verification*
    - *Code Coverage and Continuous Integration*
    - *Software Lifecycle with an Example. Community Impact*
    - *An Introduction to Software Licensing*

# More resources

- **Software Carpentry:** <http://software-carpentry.org>
  - Since 1998, Software Carpentry has been teaching researchers in science, engineering, medicine, and related disciplines the computing skills they need to get more done in less time and with less pain.
  - Lessons: <https://software-carpentry.org/lessons/>
    - freely reusable under the Creative Commons Attribution license
- **Software Sustainability Institute:** <http://www.software.ac.uk>
  - UK national facility for cultivating and improving research software to support world-class research
  - Guides: <https://www.software.ac.uk/resources/guides-everything>
- **Computational Sci. Stack Exchange:** <https://SciComp.StackExchange.com>
  - Question and answer site for scientists using computers to solve scientific problems

# Agenda

Time	Topic	Speaker
1:30pm-2:15pm	Why effective software practices are essential for CSE projects	Anshu Dubey, ANL
2:15pm-2:45pm	Better (small) scientific software teams	Michael A. Heroux, SNL
2:45pm-3:00pm	Improving Reproducibility Through Better Software Practices	Michael A. Heroux, SNL
3:00pm-3:30pm	Break	
3:30pm-4:15pm	Testing HPC Scientific Software: Introduction	Jared O'Neal, ANL
4:15pm-4:45pm	Verification, and Evaluating Project Testing Needs	Anshu Dubey, ANL
4:45pm-5:00pm	Code Coverage and CI	Jared O'Neal, ANL