

# Software Productivity Challenges in Climate Modeling

Philip Jones  
Los Alamos National Laboratory  
[pwjones@lanl.gov](mailto:pwjones@lanl.gov)

## Abstract

Climate models, like other multi-scale and multi-physics codes, incorporate a large number of software components from a wide pool of developers representing a range of scientific and software expertise. Productivity is inhibited by the need for performance portability, bottlenecks associated with community development, weaknesses in software process and an incomplete testing regime. We describe here the background of this current state and the software productivity needs for a path forward.

## Climate Models

Climate models are computer models of the Earth's climate system built from a number of large components (atmosphere, ocean, land surface, sea-ice and land-ice) that are in turn built from subcomponents representing dynamical, physical or biological processes (multi-physics). Some of these processes are unresolved and are represented by parameterizations that attempt to capture behavior at resolved time and space scales (multi-scale). To improve fidelity, climate modelers continue to push toward the highest resolution achievable, add missing processes or improve the physics already represented. Because the range of processes is very diverse, climate models are developed by a large community of scientists to capture the expertise required; as an example, the Community Earth System Model (CESM) includes contributions from hundreds of developers across the international climate community. Climate models are the only tools we have to project future climate change and climate model output is used to inform decisions on future adaptation and mitigation strategies. The use of climate models in such a context requires a strong focus on verification and validation (V&V).

## V&V Challenges

Because V&V is important for both the decision-support functions and to improve the scientific validity of the model, climate modelers have developed testing and validation workflows. Software testing includes frequent (nightly) regression and build testing, tests for exact restarts and a requirement of bit-for-bit reproducibility for changes that are not expected to result in different answers. In addition, many of the infrastructure routines have been unit tested, though not all of these unit tests are revisited since much of the infrastructure has not changed over the past several years. Much of the testing above relies on customized scripts without a comprehensive test harness or software testing toolkit. At the system scale, model validation is extremely important and we rely on extensive model-data comparisons and large multi-model intercomparison projects, both at the single component and fully coupled system levels. More recently, uncertainty quantification (UQ) techniques have begun to be implemented. In between these end points, there is

still a significant fraction of the code base that is inadequately tested, due to a number of difficult challenges. In many cases, testing has a qualitative component that inhibits automated testing. For example, a transport/advection algorithm can be tested with a number of standard experiments, but a quantitative metric is difficult because we are balancing diffusive vs. dispersive error. Testing of other process parameterization is hampered by lack of detailed observational data and in some cases, no first-principles theory that can be used to validate. Quantitative analytical tests are extremely rare. Lack of testing for these modules has been responsible for some recent errors that invalidated a number of simulations with related loss of time and computing resources. More comprehensive testing is needed with coverage across all modules, but this will require significant work in defining appropriate tests in the absence of data or reference results.

### **Software Process**

As large community code efforts like the CESM have evolved, a community development process has also emerged. Currently, CESM relies on a group of gatekeepers and software engineers for the major components and overall system. This was a choice made primarily for quality control reasons and to provide some software expertise to the science community. However, as the project has grown, this structure has not been scalable. The workload of the software engineering group is becoming unsustainable. A legacy of earlier heavyweight software engineering approaches from a decade ago also led the community to view this group as a way to shield scientists from the perceived burdens of software processes, leading to a general erosion of software expertise amongst the broader group of scientific developers. A more scalable approach, broader training in modern software practices (eg Agile, TDD) and approaches that encourage or enforce such practice are needed.

### **Performance Portability**

As climate model resolution and model complexity continue to increase, climate science will continue to require the largest capacity machines and most advanced architectures available. These new architectures are increasing in complexity as vendors respond to heat/power limitations. Performance optimization for climate models is further inhibited because model performance is evenly distributed; significant changes in performance will require changes to data structures to optimize memory access and other modifications to expose more parallelism that will need to be propagated throughout the few-million-line code base. The current architecture transition is likely to last many years as new hardware designs and programming models are explored and any convergence is achieved. Performance portability and rapid architectural change requires the development of appropriate abstractions or other techniques to permit further science development while minimizing the impact of these changes. Common libraries or frameworks may also help, but the climate community is often wary of this approach due to long-term support and the inability to debug or validate results from library routines. Performance optimization will require new software design, legacy refactoring, rapid prototyping of new ideas and V&V to test the changes along the way.