Assuring the Future?
A Look at Validating Climate Model Software

Forrest Shull

If the computer scientist is a toolsmith, and if our delight is to fashion power tools and amplifiers for minds, we must partner with those who will use our tools, those whose intelligences we hope to amplify.

—Frederick P. Brooks, Jr., The Computer Scientist as Toolsmith II

ONE OF THE attractions of the software engineering profession to me has always been the opportunities it offers to work with specialists in a wide variety of domains. This issue of IEEE Software focuses on a unique and particularly interesting domain: software built to support climate studies. This field requires the collaboration of experts across many different areas of study and whose results bear on matters of direct importance to all of us. Here in the US, the field of climate studies seems unable to stay out of the news for very long, especially in a political season such as this one. As just one example, see the recent discussion in The Washington Post regarding presidential candidates and their stands on climate change.1

What is day-to-day life like for a developer of the climate modeling software that supports such important but controversial results? To find out, I interviewed two professionals in the field who have been thinking about what it means to develop and communicate about such results: Robert Jacob, a computational climate scientist in the Mathematics and Computer Science Division of Argonne National Laboratory, and Gavin Schmidt, a climatologist and climate modeler at the NASA Goddard Institute for Space Studies.

An important theme both interviewees reiterated is that they’re certainly not interested in software for software’s sake. Rather, they’re constructing large, sophisticated, finely grained scientific tools—which happen to require software. As useful as the software and software engineering practice is, neither can be allowed to, in Robert’s words, “get in the way of the science.”

Yet these are still software projects—moreover, projects developing incredibly complicated code and requiring large, distributed teams. Even on massively parallel supercomputers, executing a climate simulation long enough to obtain a meaningful answer can take any length of time, from overnight (for testing the algorithms responsible for one or two key factors) to six months (for running experiments that predict global behavior). The development teams themselves can consist of dozens of developers, who might have some software background but typically have expertise in fluid dynamics, atmospheric science, oceanography, and many other domains. In order

---

IEEE Software Mission Statement:
To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.
to manage this complexity, the typical development team in this domain has adopted software engineering practices with concrete and short-term payoffs—configuration management, regression testing, bug trackers, and so on—even though it may have been a late adopter.

Validating Model Results

Although our conversations were wide-ranging, I steered my interviewees toward the question of how the field assesses the quality and trustability of results coming out of the software models—in other words, how does anybody know that the software is working correctly? I’ve been particularly fascinated by this issue for some time. Unlike many other domains in which software is created, in climate studies, the correct answer or behavior is not known in advance; that’s why we need the software in the first place.

Our conversation focused on a number of ways the community validates climate code. Some of these are strategies that any developer should recognize, while others are uniquely fitted to this domain.

Testing the Code

Both Robert and Gavin emphasized that climate code is typically very well tested. Both mentioned that good testing practices, such as regression testing and tracking issues to completion, are typically adopted. Nightly builds and automated test suites catch a lot of inadvertent changes.

Divide-and-Conquer Strategies

At a high level, climate models tend to have four major components that model the atmosphere, the ocean, sea ice, and land. For purposes of validation, the components can be separated and their output judged independently by experts in the relevant domain. Each of these comparisons is a large undertaking that can occupy a research scientist for years. But, as Robert suggests, this approach represents a “step-by-step approach to building confidence in the pieces.”

Surprisingly (at least to me), Robert also emphasized that the “physics behind the algorithms is pretty basic, 19th-century stuff.” Hence, validating the underlying components is feasible and an important step to assuring the larger model. However, scaling these pieces up and combining them across such complicated systems is not straightforward. As others have amply demonstrated across many domains, having confidence in software components can’t automatically translate into having confidence in the model that composes them.

Comparing Model Outputs to Analytically Derived Answers

Robert described situations where a model’s output can be validated not against the messy real world but in thought experiments where scientists can contrive conditions for which they do know the correct answer. A simple case might involve modeling atmospheric behavior on a simulated planet with no mountains. Scientists can get the right answer analytically and use that to sanity-check what the models come up with.

Benchmarking

There are international benchmarking efforts that can assess model predictions against known results—after all, we do know what’s been happening to climate in the 20th century. Gavin noted, however, that these can be a double-edged sword: benchmarks can be useful, but too much reliance on them can run the risk of overfitting the data to one dataset. Or more generally, excessive reliance on benchmarks runs the risk of biasing models toward previous conditions that might change over time. Good models in this case need to have the fundamentals right so that they can make future predictions.
Competition among Models

Multiple teams are working on independent climate models and trying to replicate the climate effects most accurately. Both Robert and Gavin emphasized that for an effect to be taken seriously, it should be reproducible across these different models, which are all in friendly competition. The fact that the models are developed by independent developers, using independent approaches, hopefully means that they will incorporate independent biases or other sources of software faults. The aim is to gain confidence that the observed results are true—that they aren’t the result of the biases and mistakes of a team or even a few teams.

As Gavin and I discussed, this is actually an example where some of the conventional best practices in software engineering don’t apply very well to this domain. Many software engineering teams invest in making basic functionality reusable so that other teams can make use of the development and validation work that’s already been invested. In this domain, however, there are situations where having teams “re-invent the wheel” is preferable to encoding a set of bugs or biases in a particular component and thus replicating those problems across independent teams. Gavin also pointed out that disagreements among models are not necessarily a bad thing—the spread of model results can be a good proxy for measuring the uncertainty of scientists’ understanding of the phenomenon.

Empirical Validation of Prediction Accuracy

The gold standard of model quality must be whether the predictions are accurate. Here, the fact that the models are continually looking into the future should not blind us to the fact that versions of these same models...
have been applied in the recent past to make predictions for events that have already come to pass. Gavin pointed me to a number of model predictions that have since been validated. For example, when Mount Pinatubo erupted in 1991, simulations predicted global temperatures would drop by around 0.5 degrees Celsius over the following two years; this matched reality very well. Likewise, the predicted effects of greenhouse gas emissions have been good matches to model predictions going back to the 1980s.

However, Robert also pointed out that the gathering of empirical data for feeding the models and comparing their outputs to reality isn’t a trivial task. Gathering the data can require getting actual readings from across an area of several hundred square miles, which can be a daunting research task in its own right.

A Lesson for Us
This list of strategies is clearly not orthogonal, and makes no attempt to be complete. But it points nevertheless to a whole toolkit of approaches used to get some confidence in the correctness of the climate simulation code. I believe it’s a worthwhile endeavor for those of us in other domains to consider how robust is the set of methods we’re using to assess the correctness of our own code.

Better Communication
Both Robert and Gavin have been dealing lately with the question of how climate scientists can communicate better regarding their work, given that elements of the public (who would be directly affected by some of these results) remain skeptical about both the findings and the process that produced them.

There has been some experimentation already with teams making their chance to publish a peer-reviewed version of the paper in IEEE Software.

The IEEE Software Best Research Paper Award at Agile 2011 went to “Decision Making in Agile Development: A Focus Group Study of Decisions and Obstacles” by Meghann Drury, Kieran Conboy, and Ken Power (see Figure A). The paper describes a focus group conducted at the Agile 2010 workshop with 43 agile developers and managers about what decisions were made at different points in the agile development iteration cycles. The paper addresses a significant challenge for software practitioners, and using a focus group as the research method is also a novel approach in studies of agile software development.

The winner for the Insights Stage was “Lean as a Scrum Troubleshooter” by Carsten Jakobsen and Tom Poppendieck. Our editorial board member Linda Rising reports that she was especially happy about this choice because she shepherded this paper. Linda describes “shepherding” as a special kind of review process where the reviewer is almost a coauthor who offers positive suggestions for improvement with the goal of helping the author(s) produce the best possible paper. The shepherd is “on the author’s side” and works toward the goal of getting the work accepted. It represents a real partnership between the author(s) and shepherd, and getting it incorporated as part of the process for Agile 2011 Conference experience reports has been due to the efforts of Linda, Rebecca Wirfs-Brock, and other conference organizers. Initially, 64 proposals were submitted to the Insights Stage. From that group, 28 were selected for shepherding, and out of that group, 26 were selected for the conference program. Conference shepherds nominated eight candidates for the best experience report, then narrowed the list down to three; our award winner was voted on by the larger committee.

We’re grateful to all those who participated on the award selection committee: Laurie Williams (chair), Nanette Brown, Torgeir Dingsøyr, Nils Brede Moe, Maurizio Morisio, Linda Rising, Helen Sharp, and Rebecca Wirfs-Brock.
code available; a Point/Counterpoint article elsewhere in this issue debates the merits of that approach head-on. But it’s certainly not the entire answer. Gavin feels that what might need to be communicated are the basic scientific principles—for example, how models support the scientific process of reasoning—not how a particular piece of software handles a particular variable or distributes a particular algorithm over multiple processors.

Robert also pointed out that the reward structure in this domain is based on how many scientific papers get published. The software development aspect is just as important, but not as well recognized. So it will be interesting to observe what outcomes will result from ongoing efforts

- to make the code more visible;
- to make the use of the code by independent scientists easier; and
- to complete the feedback loop, so that code developers have more visibility themselves into when their code has been downloaded and when it’s actually been used for something.

After all, the correctness of the code and the predictions it supports are of huge interest to all of us, whatever domain we’re working in.

Reference