Engineering the Software for Understanding Climate Change

Climate scientists build large, complex simulations with little or no software engineering training—and don't readily adopt the latest software engineering tools and techniques. This ethnographic study of climate scientists shows that their culture and practices share many features of agile and open source projects, but with highly customized software validation and verification techniques.

Computational scientists develop software in a very different way from the development processes commonly described in the software engineering literature. They build software to explore scientific questions for which the answers aren't known in advance. As a result, generic software development processes are a poor fit: it's hard to specify what software will be needed, hard to predict how long it will take to develop, and hard to verify correctness.[1] Nonetheless, the effort needed to develop and verify the code can be a bottleneck in scientific productivity. In fact, because advances in processing speed haven't been matched by advances in software development techniques, time-to-solution in many cases is growing, rather than shrinking.[2]

In this article, we describe a detailed case study of climate scientists' software development practices at a large government-funded research lab, the UK Meteorological Office's Hadley Centre for Climate Prediction and Research. Software development for climate models is interesting for numerous reasons. Advances in climate science will be central to improving our understanding of the likely impacts of climate change over this century and hence will guide government policy-making. Computational models have always played a central role in climate science, driving both a heavy demand for supercomputing power and a need for expertise in computational techniques. Any opportunities for improvements in software development practices are therefore likely to have a big impact on the field.

Our goal in this study was to investigate how scientists get their ideas into working code and reason about its correctness. We concentrated on how climate scientists' practices differ from other forms of software engineering and how they themselves view the activities around model building.

1521-9615/09/$26.00 © 2009 IEEE
Copublished by the IEEE CS and the AIP

Steve M. Easterbrook
University of Toronto
Timothy C. Johns
Hadley Centre for Climate Prediction and Research

Scientists and Software Practices: General Characteristics

As many researchers have noted, computational scientists' software development practices have several distinguishing characteristics.[1,3] Developers are trained primarily in their scientific discipline, rather than computing or software engineering, and the distinction between developers and users is blurred. The computational models are continually reworked over years or decades, so they tend to use older programming languages for which the latest software development tools are not available. Scientists have additional requirements for managing scientific code, including that they need to keep track of exactly which code version was used in a particular experiment, rerun experiments with precisely repeatable results, and build alternative versions of software for different kinds of experiments.[4] For all these reasons, scientific teams tend to develop their own tools in-house, rather than rely on external providers.

Because they generally don't know the requirements up front, computational scientists generally adopt an agile development approach. However, they don't use standard agile processes.[1] Because they focus on scientific goals rather than software quality goals, they use measures of scientific progress rather than code metrics to manage their projects.[5] Perhaps surprisingly, in the high-performance computing domain, software performance is not always the most important nonfunctional requirement—it often takes second place to code maintainability and portability.[5]

Software verification and validation (V&V) is challenging in computational science because of the lack of suitable test oracles and observational data.[3] In the Earth sciences, model validation is a major challenge[6] because a model's value as a scientific instrument doesn't always depend on its degree of isomorphism with the physical world. Sometimes, the best way to improve our understanding of Earth systems is to build a model that is unlike the Earth in some interesting way.[6] In climate modeling, scientists are faced with many choices about when to use simplifying assumptions—such as parameterizing a physical process rather than modeling it explicitly—and when to attempt to improve realism.

Finally, scientific software tends to have a very long lifetime, during which it continually evolves to reflect advances in both the science itself and the computational techniques used in the models. Hence, we can make useful comparisons with previous studies of software evolution,[7] especially those in open source[8] and agile[9] projects.

Case Study Background

We conducted an eight-week observational study at the Met Office Hadley Centre using ethnographic techniques to identify the concepts and work practices that climate scientists use, as well as to understand their perspectives. We selected the Hadley Centre because it's recognized internationally as a leader in climate modeling, has a reputation for good software development processes, and already has state-of-the-art code management practices in place.[4]

Methodology

We conducted 24 semistructured interviews with scientists across the organization and observed many meetings, including project team meetings, planning meetings, workshops, and scientific seminars. We also visited two external partner organizations to gain additional perspective on collaborative relationships. We analyzed project documentation and electronic media, including the organizational wiki and newsgroups. Finally, we extracted quantitative data from the code repository to reconstruct the software's historical evolution.

We began the study with five key research questions:

- *Correctness*: How do scientists assess the correctness of their code? What does correctness mean to them?
- *Reproducibility*: How do scientists ensure experiments can be reproduced (such as for peer review)?
- *Shared understanding*: How do scientists develop and maintain a shared understanding of the large, complex codes they use? For example, what forms of external representation do they use when talking about their models?
- *Prioritization*: How do scientists prioritize their requirements? For example, how do they find a balance between doing what is computationally feasible and what is scientifically interesting?
- *Debugging*: How do scientists detect (and/or prevent) software errors?

We investigated these questions using an ethnographic approach, focusing on aspects of the culture and practices that seemed interesting, and exploring how the scientists themselves view their work.

The Met Office Hadley Centre

The Met Office, based in Exeter, is an operational weather forecasting center that provides services to a range of customers including broadcast and print media, civil aviation, and the UK military. It employs more than 1,700 people.

The Met Office's work on climate modeling began in the early 1970s. In 1990, the Hadley Centre was created to act as a center of excellence for climate change research. It currently employs approximately 180 scientists and 19 IT specialists. Research funding is largely from government grants and EU contracts. In recent years, it has expanded its mission to include consultancy on climate impact assessments.

Climate research (CR) at the Hadley Centre is closely tied with the Met Office's larger

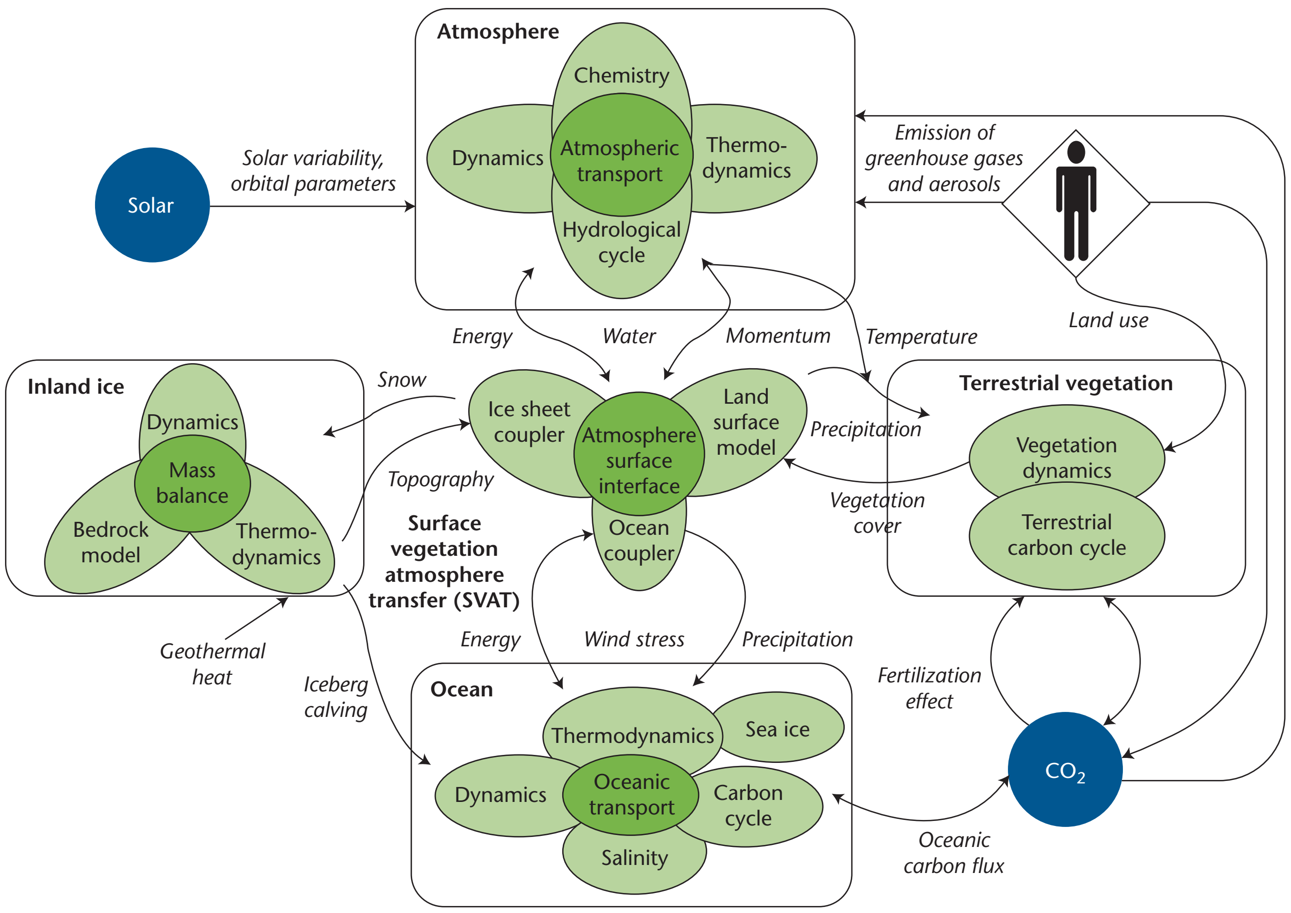

Figure 1. Conceptual view of the components and couplings of a coupled Earth system model.

meteorological research and development (Met R&D) effort. The two groups primarily occupy a single, open-plan office space on the second floor of the Met Office's Exeter headquarters and have built a single unified code base. This close relationship with an operational weather forecasting center is unusual for a climate modeling group, as is use of a unified code base. The results we present here focus primarily on climate research, but our interviews—and the development practices we observed—cover both groups.

Climate Modeling Basics

Climate scientists use a range of computational models in their research. The most sophisticated are general circulation models (GCMs), which represent the atmosphere and oceans using a 3D grid and solve the equations for fluid motion to calculate energy transfer between grid points. GCMs are designed so that the various subsystems (atmosphere, ocean, ice sheets, vegetation, and so on) can run either independently or coupled, with a coupler handling energy and mass transfers between subsystems (see Figure 1). Researchers can run the models at different resolutions, depending on the available computing power. Coarse-resolution GCMs can simulate large-scale phenomena, such as midlatitude weather systems, while finer-resolution models are needed to simulate smaller-scale phenomena, such as tropical cyclones.[10]

Scientists make many trade-offs when building climate models. It's not computationally feasible to simulate all relevant climate processes (to the level they're currently understood), so climate scientists must decide which processes to resolve explicitly and which to parameterize. They develop parameter schemes from observational data or from uncoupled runs of models that do resolve the phenomena. For example, they can use a separate cloud-resolving model to generate aggregate cloud formation data for use as GCM parameters. Judgment is needed to determine which processes and resolutions are relevant to a given research question.

The Earth's climate is a complex system, exhibiting chaotic behavior. The models might fail to match the observational data for several reasons[11]:

- measurement error (observations might contain inaccuracies),
- natural variability introduces noise to both model and observations,
- scaling/aggregation issues (for example, observation locations might not match the grid points in the model), and
- model imperfections.

It can often be hard to identify which of these factors is relevant. To investigate the latter three points, climate scientists increasingly use various types of model ensemble, including

- multimodel ensembles, to compare models developed at different labs on a common scenario;
- multimodel ensembles using variants of a single model, to compare different schemes for the model's parts (such as different radiation schemes);
- perturbed physics ensembles, to explore probabilities of different outcomes in response to systematically varying physical parameters in a single model; and
- varied initial conditions within a single model, to test the model's robustness and better quantify probabilities for predicted climate change signals.

Current issues in climate research include quantifying uncertainty, assessing the impact of climate change (such as on the occurrence of severe weather events), and producing better regional predictions.

Study Observations

We'll first describe the models developed at the Met Office and the processes by which they are developed, and then compare these processes with other software engineering projects.

The Met Office Unified Model

The Met Office maintains a common suite of Fortran routines (the Unified Model) for its numerical weather prediction and climate models. This code base has, arguably, been continually evolving for at least 30 years. The NWP and climate codes were unified about 20 years ago. Operational weather forecasting models built from the UM include a global model, a European regional model, and an ocean wave model. The climate models include HadCM3 and HadGEM1, which provided data for the Intergovernmental Panel of Climate Change's 2007 assessment;[10] HadGEM2, a newer generation of the global environment model (to be used for the next IPCC assessment); and HadGEM3, a new research model.

Most of the code was developed in-house at a single Met Office location. However, the range of expertise needed to develop climate models has grown, and it's now hard to provide all the necessary expertise in-house. Over the past few years, the Met Office has participated in several consortium efforts that complement its in-house expertise. These have led to the inclusion of the UK atmospheric chemistry model (UKCA), developed by a group of academic research labs, and the Nucleus for European Modelling of the Ocean (Nemo), a state-of-the-art model developed at the Centre National de la Recherche Scientifique (CNRS) in Paris.

The UM's current release is about 830,000 lines of Fortran. Figure 2 shows the UM's growth over the past 15 years. Discontinuities in the growth curve represent the replacement of major components: the dynamical core at version 5.0 and the ocean model at version 7.0. At version 6.3, the Met Office adopted a new code management system, flexible configuration management,[4] and cleaned up the file structure (note the deliberately faster release cycle after FCM's adoption).

Interestingly, the time taken to perform a climate run hasn't changed over UM's life because climate scientists take advantage of increases in supercomputer power to increase their models' resolution and complexity. A century-long climate simulation typically takes a couple of months to run on an NEC SX-8. Scientists more often run the models for just one to two decades of simulation—which can still take a couple of weeks, depending on the model configuration. Older models can now be run on desktop machines, but much of the leading-edge science uses the newest, higher-resolution models, which means that supercomputer capacity is a major resource constraint.

In addition to the UM, the Met Office maintains a number of other critical software systems, including the UMUI, a user interface for configuring model runs; an ancillary file generator, which takes observational datasets and creates input files for the models; and a suite of data analysis and graphics packages for studying the model outputs.

Software Evolution

As Figure 2 shows, the UM has undergone steady evolution throughout its history. Drivers for

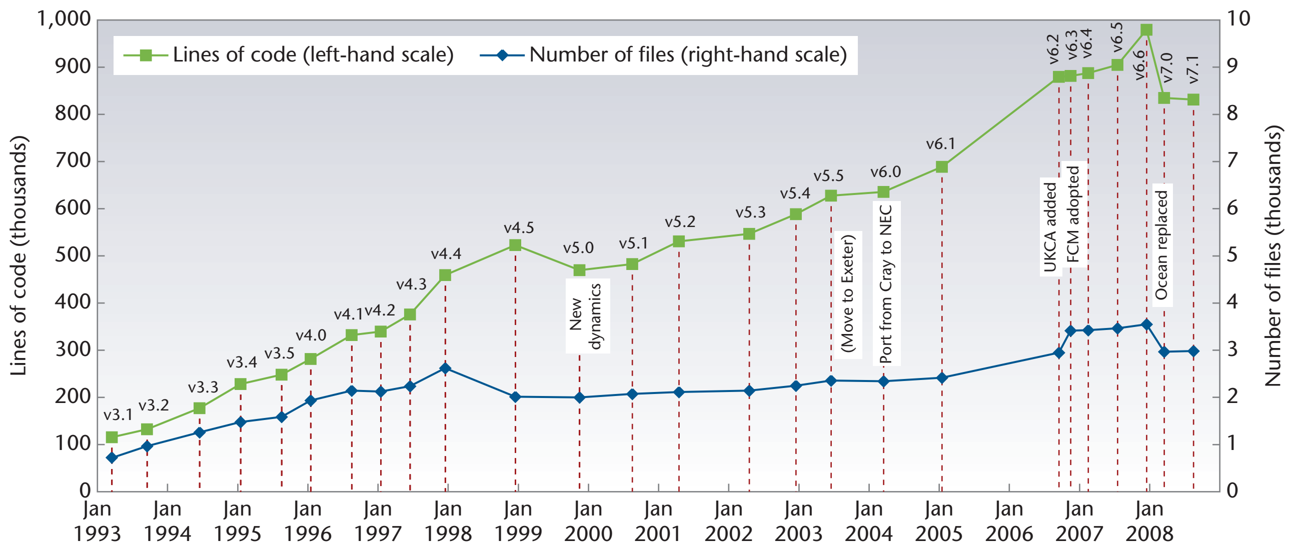

Figure 2. The Unified Model's growth over the past 15 years. Discontinuities in the growth curve indicate major component replacements, such as at versions 5.0 and 7.0, when the UM's dynamical core and ocean model were replaced, respectively. UM adopted the UK atmospheric chemistry model (UKCA) at version 6.2 and flexible configuration management (FCM) at version 6.3.

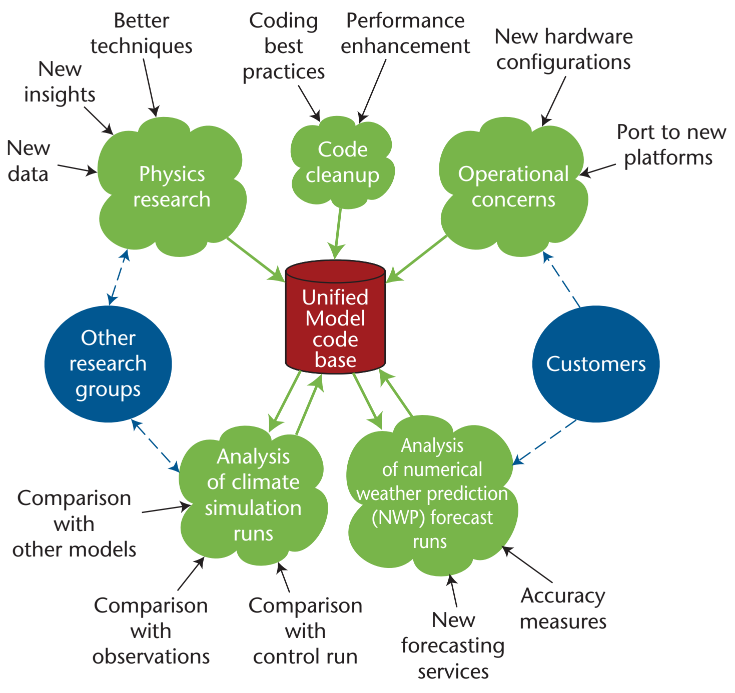

Figure 3. Drivers of change in UM code. The model's steady evolution is influenced by many factors, from advances in how scientists represent physical processes in the model to operational concerns, such as hardware upgrades. *(Courtesy of Damian Wilson, UK Meteorological Office.)*

change come from several directions (see Figure 3). First, advances in the underlying science lead to improvements in how scientists represent physical processes in the model. Second, the accuracy of the operational weather forecast runs is analyzed regularly, and systematic errors in these forecasts are investigated and corrected (where possible). Third, the outputs from climate model runs are continually compared with observational datasets and with runs of models from other centers to identify areas of weakness. Fourth, operational concerns sometimes lead to changes—for example, to allow the models to run on newer hardware.

Occasionally, scientists make opportunistic changes to the UM, to improve speed or to tidy up the code base. Such changes tend to be treated as lower priority, except in cases where they'll likely lead to major improvements to the operational forecasting models. The prevailing culture discourages such changes, in part because a scientist proposing them must demonstrate that the changes won't negatively affect the accuracy and performance of any of the operational models.

These change drivers are largely internal to the Met Office. The Met Office's customers don't interact with the software directly, so customer requirements affect the change process only indirectly—such as when forecasters identify a demand for new types of weather forecast data. On the other hand, other climate research centers do run the models; change requests from peer institutes and science review committees are filtered through Met Office contacts.

These change drivers lead to requirements conflicts. For example, there is often a trade-off between improving the models' skill in reproducing observed weather and climate variations versus improving the physics schemes' scientific validity. The models must be tuned empirically by adjusting the parameterization schemes because both the models and the parameterizations are approximations of the real physical processes. As the scientists themselves note, models sometimes get a good match with observations for the wrong reasons. (This problem resembles the overfitting sometimes observed in benchmarking.)

This conflict is complicated by the use of the UM for weather forecasting, which places constraints on performance (forecasts must be delivered on time) and accuracy (the Met Office has annual targets for improvements in weather forecast accuracy). Scientific improvements often have negative effects on performance or forecast accuracy, so they might be delayed until they can be included in a bundle of changes that has an overall positive effect. In addition, alternative versions of various physics schemes are included in the UM, which can be selected when different models are built.

The Development Process

The Met Office's software development processes have also undergone significant evolution over the UM's life. In particular, the FCM adoption has institutionalized practices that previously were not applied systematically. Here, we describe the processes we observed in the summer of 2008, two years after FCM was introduced.

Met Office staff play a number of distinct roles, organized like the "onion" model often observed in open source projects. At the core, approximately 12 people from the two IT support teams (Met R&D and CR) control the acceptance of changes into the UM's trunk. They act as experts for integration and platform-specific issues. Many of them have scientific backgrounds, with PhDs in numerical computing or related fields. At the next layer, about 20 of the more senior scientists act as code owners, each responsible for specific UM sections (such as atmosphere, ocean, boundary layer, dynamical core, and so on). Code owners are domain experts who keep up to date with the relevant science and maintain oversight of developments to their sections of the model. Membership in these two layers rarely changes.

In the outer layers are scientists who run the models as part of their research. A configuration manager is appointed for each climate model (usually a more junior scientist). Configuration managers become local experts for knowledge about how to configure the model and track experiments performed with the model. Approximately 100 Met Office scientists contribute code changes for any given release. Finally, a broader group of scientists both within and outside the Met Office make occasional use of the models and might suggest potential improvements.

UM releases are planned on a regular schedule, typically every four to five months. All model changes are captured as FCM tickets, which are allocated to upcoming releases using an agile-planning approach. Approximately three months into the release cycle, there's a deadline for tickets to be included for that release, and one month later there's a code freeze—the IT teams refer to it as a "frosting" rather than a freeze, as some changes are still permitted. One further month is allowed for the IT teams to ensure all model configurations work properly, fix any remaining bugs, and ensure the UMUI is updated. A release date isn't fixed until this work is complete.

Each change passes through two review stages before being accepted into the UM's trunk. First, the relevant code owner runs a scientific review. Significant changes are typically discussed with code owners in advance to explore the scientific justification and relative priority; smaller changes are submitted for review once they're complete. Second, an IT team member performs a system review. This focuses on coding standards, basic code hygiene (for example, to verify that files are opened and closed properly), potential performance issues, and integration testing across different model configurations. Once the review is passed, the IT teams accept the changes (no more than four per day) into the trunk and run an automated test harness every night on the updated trunk. Tickets are closed once the changes pass this overnight test.

The process is overseen with lightweight project management. Each named climate model under active development is explicitly defined as a project to identify the strategic scientific objectives, allocate resources, and manage risk. Within a project, however, a bottom-up strategy dominates, with individual team members taking the initiative to identify what needs doing and to prioritize tasks. The result is a hybrid strategy in which code development proceeds using an agile-planning approach, while specific model configurations are more carefully controlled.

Verification and Validation

V&V processes are dominated by the understanding that the models are imperfect representations of highly complex physical phenomena. Instead of reasoning about "code correctness," Met Office scientists treat the models as evolving theories

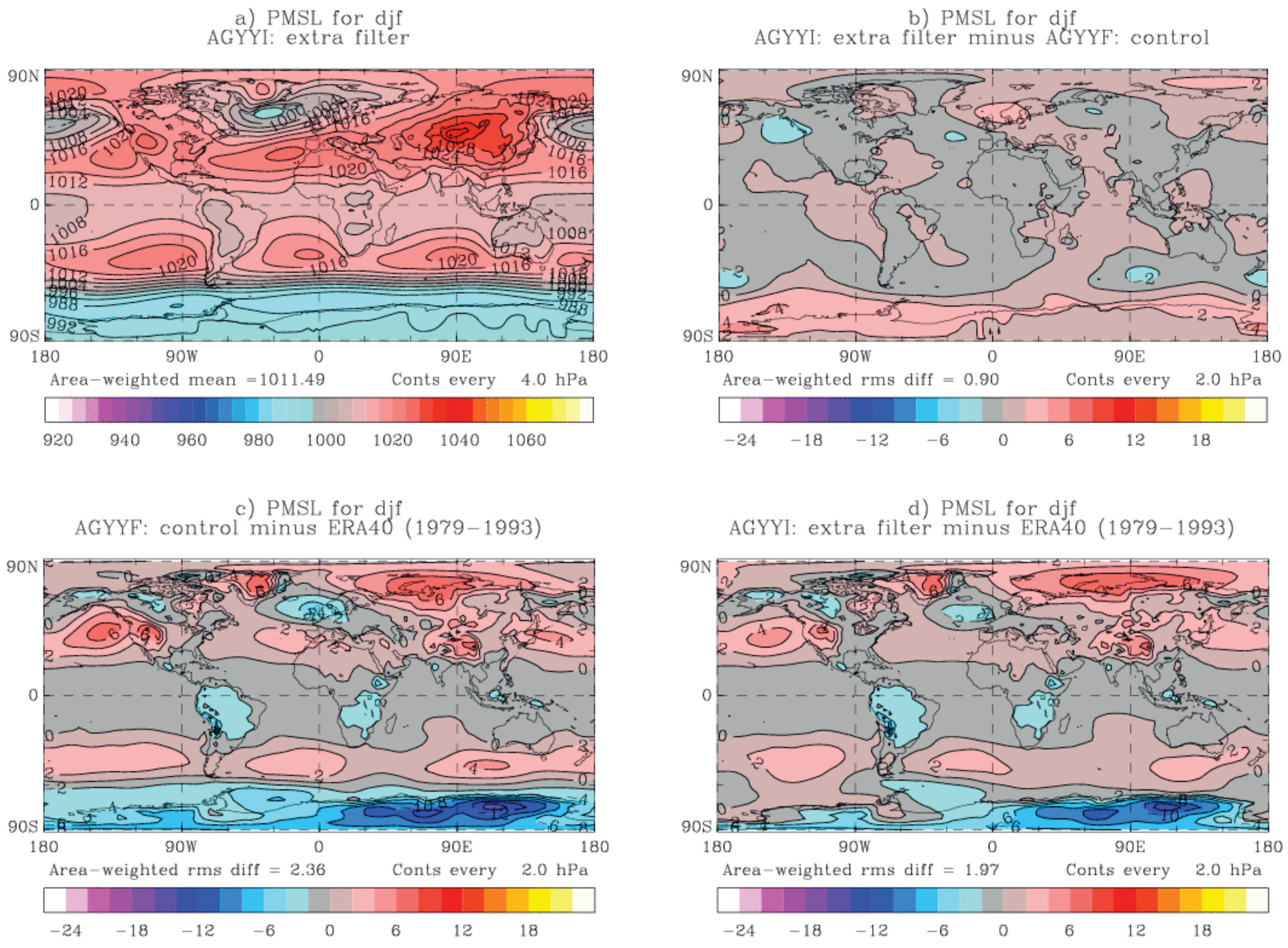
a) PMSL for djf
AGYYI: extra filter
Area−weighted mean =1011.49
Conts every 4.0 hPa
b) PMSL for djf
AGYYI: extra filter minus AGYYF: control
Area−weighted rms diff = 0.90
Conts every 2.0 hPa
c) PMSL for djf
AGYYF: control minus ERA40 (1979−1993)
Area−weighted rms diff = 2.36
Conts every 2.0 hPa
d) PMSL for djf
AGYYI: extra filter minus ERA40 (1979−1993)
Area−weighted rms diff = 1.97
Conts every 2.0 hPa
90N
0
90S
180
90W
0
90E
180

Figure 4. An example validation note. This note shows the effect of a new polar filter on mean pressure at sea level (PMSL) for December to February (djf): (a) the new model run, (b) the new run minus the control run, (c) the control run minus the observational data, and (d) the new run minus the observational data.

and conduct experiments using the models to test specific hypotheses.

Indeed, they treat each model change as an experiment. A previous run is used as a control, the changed code is the experimental condition, and observational datasets are used to assess whether the change has had the expected effect. As Figure 4 shows, automated tools, or *validation notes*, generate visualizations of selected model outputs. The scientists analyze these to understand model error and ways to reduce it. Such visualizations appear everywhere: on people's desks, pinned to the walls, passed around in meetings, and on PowerPoint slides used in scientific seminars. The example in Figure 4 shows a change that has reduced the model errors in the Antarctic.

Essentially, with this approach, the scientists are performing continuous integration testing, but they don't view it as such because for them it's part of the business of "doing science." They don't need "finished" software to perform these experiments, but they continually experiment with the software itself to improve their understanding.

A second V&V strategy is to automatically check for bit comparison between the outputs of two different runs. This is useful for checking that a change didn't break anything it shouldn't. Each change is designed so that it can be turned off (via runtime switches) to ensure that previous experiments can be reproduced. However, reproducibility can be guaranteed only if the outputs of the old and new runs are exactly identical (down to the least significant bits). Because the models are designed to run on different platform configurations, bit comparison tests can also check that all configurations give identical results. IT staff members run these tests and maintain a wiki page listing changes that break bit-level reproducibility.

This use of bit-level comparison for automated regression and configuration testing might be unique to the climate modeling community. Because full model runs take so long, a useful shortcut is to run the model for, say, one simulation day (which only takes a few minutes) and run the bit comparison test on all model variables. Bit-level reproducibility on a short run is a good indicator of reproducibility over longer runs. The overnight test harness runs many such tests. However, this practice enforces a strong conservativeness on model changes, so that refactoring is almost impossible, except when bit comparison is already lost for other reasons, such as on porting to a new supercomputer.

Other V&V strategies include formally comparing results with other models in a series of community-wide Model Intercomparison Projects[12] or, more informally, facilitating debugging.

Overall code quality is hard to assess. Model configuration problems and code defects frequently prevent an experimental change to the model from running, but these are quickly fixed. Scientists treat some errors as modeling approximations, rather than defects. For example, defects in the numerical routines that cause model drift are hard to fix, so they might be accommodated by making periodic corrections rather than by fixing the underlying routines.

The combination of the scientists' continuous integration testing and bit reproducibility tests catches most errors prior to release. The last six releases averaged about 24 bug-fix tickets per release, against an average of 50,000 source lines of code (SLOC) touched per release. This suggests that, on average, two defects per 1,000 changed SLOC make it through testing and review for each release. (We thus estimate a post-release defect density for the current release of 0.03 defects per KSLOC, or approximately 24 latent defects in 831,157 SLOC.) However, some of these bug fixes represent defects that were treated as acceptable model imperfections in previous releases. We plan to complete a more detailed analysis of post-release code defects as a followup study.

Maintaining a Shared Understanding

Met Office scientists use several different strategies to maintain a shared understanding of the software. Although there's formal design documentation for the UM, it's updated only sporadically. The scientists working on the model rely heavily on face-to-face communication, together with many "informalisms."[13]

The office's large open floorplan encourages face-to-face communication; most CR and Met R&D staff are accessible without traversing any doors or stairs. The office culture discourages noise, but many brief one-on-one technical conversations are held at people's desks. Longer conversations and meetings are held in meeting pods scattered around the office—or in social gathering spaces on the landings and the ground floor coffee shop. Several people commented that coordination has improved dramatically since the move to Exeter; previously, CR and Met R&D were in separate buildings. Cross-functional teams are often formed to investigate specific model issues.

The teams also use electronic media extensively for informal communication and coordination. A site-wide wiki serves as a repository for design notes, to-do lists, task status reports, glossaries, and so on. Scientists use site-specific newsgroups for both social interaction and technical communication, such as to broadcast the status of trunk integration, overnight test results, and problems encountered.

Representations of the code itself are rare. Occasionally, people draw flowcharts to show a new scheme's control structure. Descriptions of designs, defects, and potential improvements tend to focus on the underlying equations. Test results are described using visualizations and measurements of root mean squared (rms) error against observational data. Some scientists use the wiki as an electronic lab book, creating a page to describe each model run and its results.

When asked about the major challenges in their work, nearly everyone we interviewed mentioned the effort needed to coordinate their work with others: keeping their branches up to date, knowing what changes are happening elsewhere, managing the model configuration options, and so on. Some described coordination problems with external groups who are using older versions of the model. (The public releases of the model tend to lag the internal releases by at least a year.)

Community Models

The fact that model development has taken place at a single site appears to be important. David Randall reports that all existing GCMs were developed at large research labs, with no geographically distributed development, and he suggests that the complexity of the coupling prevents it.[14] However, occasionally a module is transplanted from one lab to another. For example, the original ocean model used in the Hadley Centre GCMs was an early version of the Modular Ocean Model (MOM) developed at the Geophysical Fluid Dynamics Laboratory (GFDL) in Princeton, New Jersey. Its replacement, Nemo, was developed at

CNRS. Such transplants allow a modeling group to tap into expertise available elsewhere.

However, it can be hard to integrate a complex, externally developed component into an existing GCM. The component might need to be ported and optimized for performance on a different computer. To get the coupling to work, both the GCM and the new component might need to be modified. Seemingly trivial technical details (such as tiny differences in physical constants) can cause problems and are hard to track down. When MOM was incorporated into the UM, the modifications led to a fork from the GFDL model, which meant that although the Met Office gained a state-of-the art ocean model, it lost access to GFDL's ongoing scientific expertise to keep the ocean model updated.

Open source teams usually dislike code forking because it divides communities and prevents the subsequent sharing of changes. The same problems occur for scientific software. To avoid this problem for Nemo, the Met Office, CNRS, and other partners have established a consortium agreement, but coordination remains a challenge. Nemo is maintained by a small group in Paris, whose development cycle doesn't match that of the Met Office. The consortium has to deal with the tension between members who want to customize Nemo for use in their coupled models and the Nemo team members who want to preserve portability and flexibility.

Increasingly, community collaborations help the Met Office foster links with other research groups and tap into pools of expertise not available in-house. Examples include UKCA; Jules, a land surface scheme; and HiGEM, a community adaptation of HadGEM1 for higher resolutions. However, in each of these community efforts, control of the core code base remains at a single site, and the partners tend to specialize in particular areas and submit their changes to the central site for inclusion in the reference model. The result is that the core site can become a bottleneck.

Discussion

Our study confirms observations of the software practices of computational scientists reported elsewhere.[1,3] The scientists have little formal training in software engineering and are skeptical of most claims for software engineering tools. However, where such tools match their needs—such as for code management and version control—they're readily adopted. The software itself has a long lifetime, is written in an "old" programming language (Fortran), and performance issues are carefully balanced with maintainability and portability concerns. As in Richard Kendall and his colleague's study,[5] we found that the developers had a strong, shared scientific background and an informal, collegial management style. The culture was one of member participation and shared responsibility.

We found no evidence of Douglass Post's slow down in time-to-solution.[2] The near-linear growth of the UM over the past 15 years indicates a steady growth of functionality, despite the model's growing complexity. This steady growth is also inconsistent with the findings of Meir Lehman and colleagues, who found that for large commercial systems, the growth rate tails off as the software increases in size and complexity.[7] Instead, it more closely resembles the evolution patterns of open source projects reported by Michael Godfrey and Qiang Tu.[8] We hypothesize that this is due to the many shared features with open source projects.

Another notable finding is the broad set of V&V approaches. The use of bit-comparison tests as a technique for regression testing appears to be unique to this community, and it reflects both a scientific concern for reproducibility of experiments and the challenges of automating testing when full runs can take weeks or months to complete. Frequent end-to-end integration testing is built into the scientific practices. Scientists spend a lot of time experimenting with each change to the model, comparing the results with control runs and observational data. Hence, integration testing is not regarded as a costly burden (even though it is costly) because it's part of doing science. The extensive use of model inter-comparisons and model ensembles is also a distinct feature of this community.

In some ways, the organization behaves like an agile software development company, with a large open office and a strong reliance on informal communication channels. It uses many agile development practices, including release planning, onsite customer, collective ownership, continuous integration, and risk management, but it doesn't use any established agile process. It also operates on a scale much larger than any agile team described in the literature.

These observations suggest interesting insights into agile practices. The Met Office has developed a set of practices that work very well for its particular context. They resemble agile practices in part because they share with them a key characteristic: over a period of time, a set of smart, engaged people have figured out for themselves what works.

This supports the argument that agile development is a set of best practices adopted by developers because they work well in a particular setting. It's also consistent with our study of successful software startup companies,[15] in which each company could be characterized as using a subset of "agile" practices, but within a distinct, homegrown process model. Over time, it appears that these organizations evolve processes that are highly adapted to their "ecological niche." This probably occurs only in stable, well-established teams with a long history of working together. Such observations suggest that domain-independent process models (such as Rational Unified Process, Scrum, XP, and so on) are simply irrelevant to these organizations.

A comparison with open source software (OSS) projects[8] is also apt:

- The Met Office's release schedule isn't driven by commercial pressure because the code is used primarily by the developers themselves, rather than released to customers.
- The developers are also the domain experts. Most have PhDs in meteorology, climatology, numerical methods, or related disciplines, and most of them regularly publish in the top peer-reviewed scientific journals.
- The code is controlled by a few code owners, with a careful review process to decide which changes are accepted.
- The community operates as a meritocracy. Roles are decided based on perceived expertise within the team. Code owners are the most knowledgeable domain experts, and code ownership tends to be stable over the long term.
- The scientists are not full-time developers. They change the model only when they need something fixed or enhanced. They don't like to delegate code development tasks to others because they have the necessary technical skills, understand what needs doing, and doing it themselves is much easier than explaining their needs to someone else.
- V&V practices rely on the fact that the developers are also the primary users and are motivated to try out each other's contributions.

However, the Met Office lacks two key distinguishing traits of OSS projects: geographically distributed teams and a commitment to open source licensing. This suggests an interesting hypothesis: the success of open source projects might have more to do with a community of domain experts building and testing software for their own use, rather than any commitment to the philosophy of free/open source software and volunteerism. Such experts figure out over time how to solve problems of coordination and communication, prefer to work in a meritocracy, and build or adapt their own tools rather than rely on commercial tools.

Limitations

We used an ethnographic approach for this study, investigating how the scientists themselves talk about their work. Mapping their concepts onto terms used in the software engineering literature might be problematic. For example, it was hard to distinguish software development from other aspects of the scientific practice, including data analysis, theorizing, and the development of observational datasets. From a scientific viewpoint, the distinction between changing the code and changing the parameters is artificial, and scientists often conflate the two—they sometimes recompile even when it shouldn't be necessary. Therefore, characterizations of model evolution based purely on source code changes miss an important part of the picture. We would need to analyze the evolution of datasets (ancillary files and input parameters) to overcome this limitation.

Similarly, there's a difference in how the scientists perceive defects and bug fixes compared to the view in software engineering literature: scientists accept model imperfections as inevitable. Hence, any measure of defect density might not be comparable with that of other types of software.

We can't claim that the observations in this case study generalize. As a climate modeling center, the Met Office Hadley Centre is unique in some ways, particularly in the close relationship with a major operational weather forecasting facility. The software development processes are highly tailored to the Met Office's needs. Hence, our observations about why such tailoring has occurred and why the processes work is likely to be more useful than any specific detail of the processes themselves.

The Met Office has evolved a mature domain-specific software development process that is highly adapted to its needs, relies heavily on the deep domain knowledge of the scientists building the software, and is tightly integrated with their scientific research practices. Model validation is extensive because it's built into a systematic integration and regression testing process, with each model run set up as a controlled experiment. The V&V practices are absorbed so thoroughly into the scientific research that the scientists don't regard them as V&V. However, the Met Office does appear to rely heavily on informal face-to-face communication to allow its scientists to develop a shared understanding of their models.

The Met Office's software development practices share many features with both agile and open source development. The comparison offers interesting insights into why the practices used by these communities work. In particular, all three communities (open source, agile, and scientific software) rely on their developers' expertise and self-organization. We hypothesize that under such circumstances, the developers will gradually evolve a set of processes that are highly customized to their context and that domain-independent process models are unlikely to work. However, further research into such comparisons is needed to investigate these observations. CiSE

Acknowledgments

We thank everyone at the Met Office who participated in this study for many stimulating discussions, and Jorge Aranda, Greg Wilson, Jeff Carver, Janice Singer, and Jon Pipitone for comments on earlier drafts. The Natural Sciences and Engineering Research Council of Canada provided primary funding for the study. Tim Johns was supported by the Defra & MoD Integrated Climate Programme (Defra) GA01101, (MoD) CBC/2B/0417 Annex C5.

References

1. J. Carver et al., "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies," *Proc. 29th Int'l Conf. Software Eng.* (ICSE 07), IEEE CS Press, 2007, pp. 550–559.
2. D. Post, "The Coming Crisis in Computational Science," keynote, IEEE Int'l Conf. High-Performance Computer Architecture: Workshop on Productivity and Performance in High-End Computing, 2004; www.tgc.com/hpcwire/hpcwireWWW/04/0319/107234.html.
3. J. Segal and C. Morris, "Developing Scientific Software," *IEEE Software*, vol. 25, no. 4, 2008, pp. 18–20.
4. D. Matthews, G.V. Wilson, and S.M. Easterbrook, "Configuration Management for Large-Scale Scientific Computing at the UK Met Office," *Computing in Science & Eng.*, vol. 10, no. 6, 2008, pp. 56–65.
5. R. Kendall et al., "Development of a Weather Forecasting Code: A Case Study," *IEEE Software*, vol. 25, no. 4, 2008, pp. 59–65.
6. N. Oreskes, K. Shrader-Frechette, and K. Belitz, "Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences," *Science*, vol. 263, no. 5147, 1994, pp. 641–646.
7. M. Lehman et al., "Metrics and Laws of Software Evolution—The Nineties View," *Proc. Int'l Software Metrics Symp.* (Metrics 97), IEEE CS Press, 1997, pp. 20–43.
8. M. Godfrey and Q. Tu, "Evolution in Open Source Software: A Case Study," *Proc. IEEE Int'l Conf. Software Maintenance* (ICSM 00), IEEE Press, 2000, pp. 131–142.
9. A. Capiluppi et al., "An Empirical Study of the Evolution of an Agile Developed Software System," *Proc. 29th Int'l Conf. Software Eng.* (ICSE 07), IEEE CS Press, 2007, pp. 511–518.
10. D.A. Randall et al., "Climate Models and Their Evaluation," *Climate Change 2007: The Physical Science Basis, Working Group I, 4th Assessment Report, Intergovt. Panel Climate Change*, S. Solomon et al., eds., Cambridge Univ. Press, 2007, p. 589.
11. R. Katz, "Techniques for Estimating Uncertainty in Climate Change Scenarios and Impact Studies," *Climate Research*, vol. 20, no. 2, 2002, pp. 167–185.
12. T. Phillips et al., "Evaluating Parameterizations in General Circulation Models: Climate Simulation Meets Weather Prediction," *Bulletin Am. Meteorological Soc.*, vol. 85, no. 12, 2004, pp. 1903–1915.
13. W. Scacchi, "Free/Open Source Software Development: Recent Research Results and Emerging Opportunities," *Proc. IEEE Int'l Symp. Foundations Software Eng.* (ESEC/FSE 07), 2007, pp. 459–468.
14. D.A. Randall, "A University Perspective on Global Climate Modeling," *Bulletin Am. Meteorological Soc.*, vol. 77, no. 11, 1996, pp. 2685–2690.
15. J. Aranda, S. Easterbrook, and G. Wilson, "Requirements in the Wild: How Small Companies Do It," *Proc. IEEE Int'l Requirements Eng. Conf.*, IEEE CS Press, 2007, pp. 39–48.

***Steve M. Easterbrook** is a professor of computer science at the University of Toronto. His research interests include software requirements analysis, software verification, and human aspects of software development, especially as applied to Earth system models. Easterbrook has a PhD in computing from Imperial College, London. He is a member of the ACM and the American Geophysical Union. Contact him at sme@cs.toronto.edu.*

***Timothy Johns** is manager of the Global Coupled Modelling team at the Met Office Hadley Centre, where he works on global climate model development, including the Unified Model and its software infrastructure, and on modeling and understanding climate change using coupled ocean-atmosphere climate models. Johns has a PhD in astrophysical modeling from University College, Cardiff. He is a fellow of the Royal Meteorological Society. Contact him at tim.johns@metoffice.gov.uk.*