Engineering the Software for Understanding Climate Change

Steve M. Easterbrook
Dept of Computer Science, U. of Toronto
Canada
sme@cs.toronto.edu

Timothy C. Johns
Met Office Hadley Centre
United Kingdom
tim.johns@metoffice.gov.uk

Abstract

Climate scientists build large, complex simulations with little or no software engineering training, and do not readily adopt the latest software engineering tools and techniques. In this paper, we describe an ethnographic study of the culture and practices of climate scientists at the Met Office Hadley Centre. The study examined how the scientists think about software correctness, how they prioritize requirements, and how they develop a shared understanding of their models. The findings show that climate scientists have developed customized techniques for verification and validation that are tightly integrated into their approach to scientific research. Their software practices share many features of both agile and open source projects, in that they rely on self-organisation of the teams, extensive use of informal communication channels, and developers who are also users and domain experts. These comparisons offer insights into why such practices work.

1. Introduction

The software development processes used by computational scientists for large, high performance software systems appear to differ significantly from the processes commonly described in the software engineering literature. Such software is typically built to explore scientific questions for which the answers are not known in advance. Traditional software development processes are a poor fit: it is hard to specify what software will be needed, hard to predict how long it will take to build, and hard to verify correctness [4].

Until recently, the computational science and software engineering communities largely ignored one another. However, there is increasing potential for the effort needed to develop and verify the code to be a bottleneck in scientific productivity [26]. Post argues that advances in computing power have not been matched by advances in software development techniques, so that time-to-solution in many cases is growing, rather than shrinking [15]. A recent DARPA HPC initiative [2] focusses on this problem by seeking improvements in programmer productivity.

In this paper, we report on a detailed case study of the practices used by climate scientists at a large government-funded research lab, the Met Office Hadley Centre. Software development for climate models is particularly interesting for a number of reasons. Advances in climate science will play a central role in shaping our understanding of the likely impacts of climate change over the next few decades, and hence will be particularly important for government policy-making. Computational models have always played a central role in this field, driving both a heavy demand for supercomputing power, and a need for expertise in computational techniques. If there are opportunities for improvements in software development practices, they are likely to have a big impact on the field. On the other hand, if current software engineering techniques do not offer any immediate benefit, this presents an interesting challenge for software engineering researchers.

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

2. Related Work

Recent studies of the software development practices of computational scientists (e.g. [4, 22, 2]) identify a number of distinguishing characteristics. The 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 developed over years or decades, and so tend to use older programming languages, for which the latest software development tools are not available. Scientists have additional requirements for managing scientific code: they need to keep track of exactly which version of the code was used in a particular experiment, they need to re-run experiments with precisely repeatable results, and they need to build al-

ternative versions of the software for different kinds of experiments [13]. For all these reasons, scientific teams tend to develop their own tools in-house, rather than relying on external providers [2].

Computational scientists generally adopt an agile philosophy, because the requirements are generally not known up front, but they do not use standard agile process models [4]. Such projects focus on scientific goals rather than software quality goals, and so use measures of scientific progress rather than code metrics to manage their projects [11]. Perhaps surprisingly, in the High Performance Computing domain, software performance is not always the most important non-functional requirement – it often takes second place to code maintainability and portability [2, 11].

These studies have also indicated a rich diversity of contexts in which scientific software is developed and used. Hence, there is now a need to characterize these contexts, to identify what is common or distinct about each of them.

Software Verification and Validation (V&V) is particularly hard in computational science [4], because of the lack of suitable test oracles and observational data. The challenge is widely recognised in the earth sciences [14, 25], where it is often argued that models should not be judged by the degree of isomorphism with the physical world, but by their usefulness as scientific instruments [24]. Oreskes argues that the appropriate use of models is for hypothesis testing and exploring "what-if" questions, rather than for predictions and scientific "proof" [14]. In climate modeling, scientists are faced with many choices about when to use simplifying assumptions (e.g. parameterizing some physical processes rather than modeling them explicitly), and when to attempt to improve realism. Hansen argues that a top-down strategy, specifying "what to compute and how to do it" does not produce the best science [9, p160].

The software development practices we describe in this case study share many attributes of open source projects. The culture of open source communities has been studied extensively in the last decade, revealing that although these communities are sometimes characterized purely by their commitment to open source licensing models, they have evolved a rich culture of software development practices, including a bottom-up planning process, careful control over code management and issue tracking, and a high degree of shared mental models, due in part to the fact that the developers are also both users and domain experts [6]. They also make effective use of informal electronic communication for knowledge sharing and consensus building [21].

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, useful comparison can be made with previous studies of software evolution [12], especially studies of evolution in open source [8] and agile [3] projects.

3. Case Study Background

3.1. Methodology

We conducted an eight-week observational study at the Met Office Hadley Centre, using ethnographic techniques to identify the concepts and work practices used by the climate scientists, and to understand their perspectives.

We selected the Hadley Centre for this study because it is recognized internationally as a leader in climate modeling, and has a reputation among climate scientists as having particularly good software development processes. This was important because we were particularly interested in best practices. Prior to this study, we had investigated the Met Office's code management practices [13]. Their use of state-of-the-art configuration management and issue tracking indicated we would be able to focus on the core scientific practices, and avoid the accidental complexity that arises from poor code management.

We conducted 24 semi-structured interviews with scientists selected from across the organisation, and observed a number of meetings, including project team meetings, planning meetings, workshops, and scientific seminars. We visited two external partner organisations, to gain additional perspectives on collaborative relationships. We analyzed project documentation and electronic media, including the organisational wiki and newsgroups. Notes from the interviews and observational sessions were typed up and cross-referenced with documentation in the electronic repositories, and we used open coding to identify common themes. Finally, we extracted quantitative data from the code repository to reconstruct the evolution history of the software.

We began the study with five key research questions:

- *Correctness*: How do scientists assess correctness of their code? What does correctness mean to them?
- *Reproducibility*: How do scientists ensure experiments can be reproduced (e.g. for peer review)?
- *Shared Understanding*: How do scientists develop and maintain a shared understanding of the large complex codes they use? E.g. 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 balance between doing what is computationally feasible and what is scientifically interesting?
- *Debugging*: How do scientists detect (and/or prevent) errors in the software?

3.2. The Met Office Hadley Centre

The Hadley Centre for Climate Prediction and Research is part of the Meteorological Office ("Met Office"), based in Exeter, UK. The Met Office is an operational weather

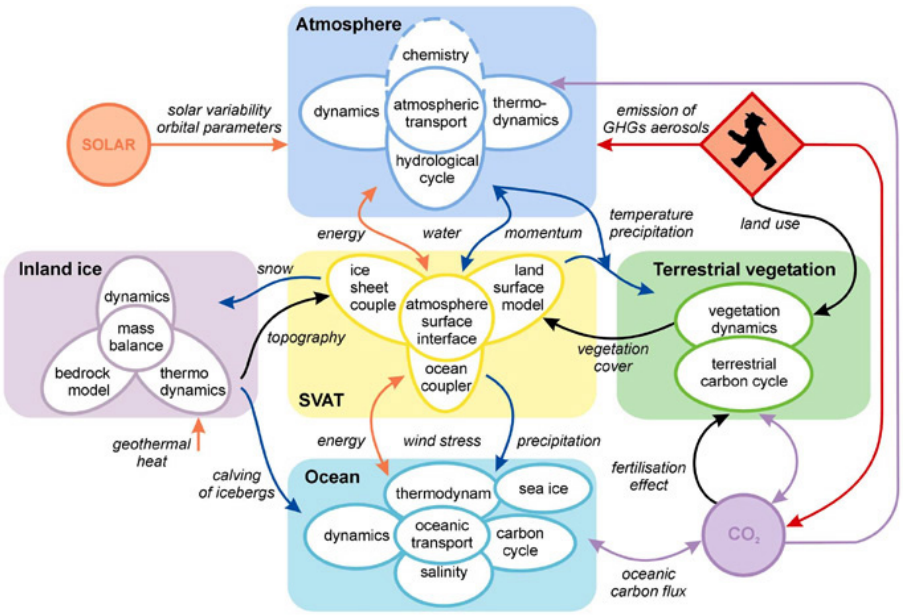

Figure 1. Conceptual view of the components and couplings of a coupled earth system model

forecasting centre, providing essential forecasting services to a range of customers including broadcast and print media, civil aviation, and the UK military. It is a Trading Fund within the Ministry of Defense (MoD), which requires it to operate on a commercial basis, and meet performance targets set by the ministry. It employs over 1700 people.

The Met Office's work on climate modeling began in the early 1970's. In 1990, the Hadley Centre was created to act as a centre of excellence for research in climate change. The Hadley Centre currently employs about 180 scientists and 19 IT specialists. Research funding is largely from government grants, through the MoD and Defra (the UK Department for Environment, Food and Rural Affairs), as well as EU contracts. In recent years, it has expanded its mission to include consultancy on climate impacts assessment.

Climate Research (CR) at the Hadley Centre is closely tied with a larger Meteorological Research and Development (Met R&D) effort at the Met Office. The two groups mostly occupy a single open plan office space on the second floor of the Met Office headquarters in Exeter, and have built a single unified code base. This close relationship with an operational weather forecasting centre is unusual for a climate modeling group, and the use of a unified code base is probably unique. This also makes it hard to study one in isolation. The results we present here focus primarily on climate research, but our interviews, and the development practices we observed cover both groups.

3.3. Basics of Climate Modeling

Climate scientists use a range of different types of computational model in their research. The most sophisticated are General Circulation Models (GCMs), which represent the atmosphere and oceans using a 3-dimensional grid, and solve the equations for fluid motion to calculate energy transfer between grid points. Such models can be run at different resolutions, depending on the available computing power. For example, a high resolution version of the Hadley Centre model being developed to run on the Japanese Earth Simulator will have an atmosphere resolution of 60 levels vertically and $0.6\,^\circ$ horizontally ($\approx$ 60km in the mid-latitudes), with an ocean resolution of $0.3\,^\circ$. Older models, with lower resolutions can be run on desktop machines.

The resolution of the model partly determines how well various atmospheric phenomena are simulated. Coarse resolution GCMs can simulate large scale phenomena such as mid-latitude weather systems, while finer resolution models are needed to simulate smaller scale phenomena such as tropical cyclones [18]

GCMs are designed so that the subsystems can run independently, or coupled together. For example, the atmosphere module can be run separately, or coupled with an ocean model, with a coupler handling transfers of energy, water and momentum at the ocean surface. Further components can be coupled to a GCM, including modules representing sea ice, atmospheric chemistry (dust and air pollution), ocean biochemistry, vegetation and carbon cycles, and so on (see Figure 1). To simulate climate change over a century, a fully coupled model can take several months to run on a supercomputer.

A number of trade-offs are made when building climate models. As it is not computationally feasible to fully represent all relevant climate processes (to the level they are currently understood), climate scientists have to decide which processes to resolve explicitly in the model, and which can

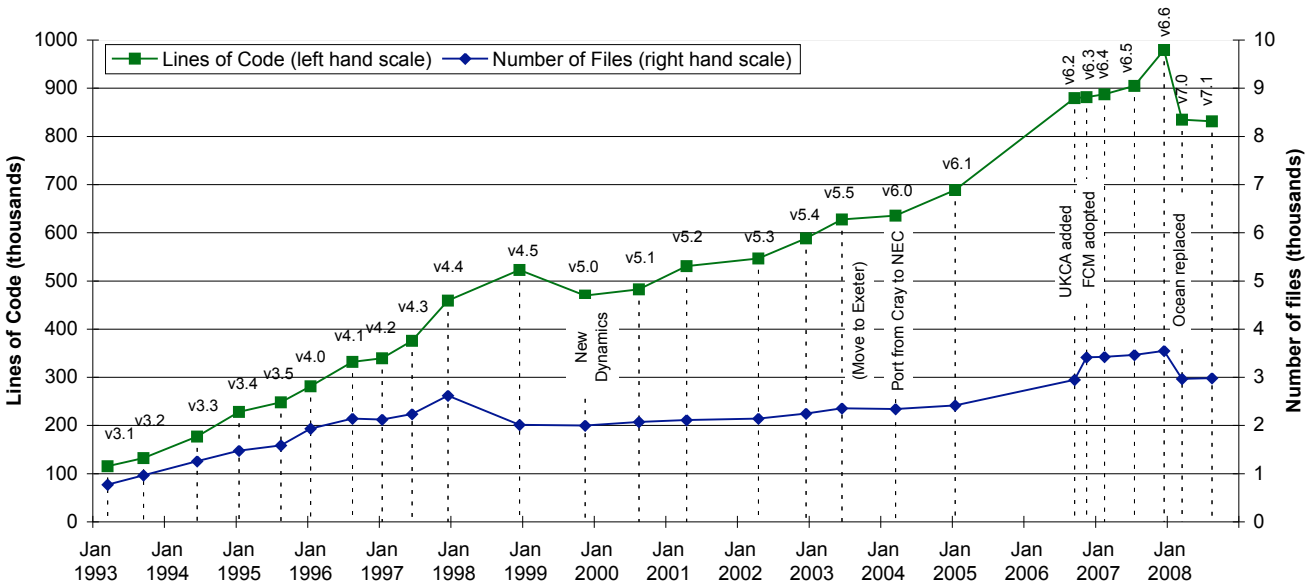

Figure 2. Growth in size of the Unified Model over the past 15 years

be represented by a set of parameters. Parameter schemes are developed from observational data, or from uncoupled runs of models that do resolve the phenomena. For example, a separate cloud resolving model can be used to generate aggregate data on cloud formation to use as parameters in a GCM. Judgment is needed about which processes are relevant at a particular scale of analysis, and which resolutions to use to study a given problem.

The Earth's climate is a complex system, exhibiting chaotic behaviour [19]. The models may fail to match the observational data for a number of reasons, and it can be hard to identify which are applicable [10]:

- Measurement error (the observations can be wrong).
- Natural variability, from small scale weather systems to the El Niño oscillation in the Pacific, introduces uncorrelated noise to both model and observations.
- Scaling/aggregation issues (e.g. the observation locations may not match the grid points in the model).
- Model imperfections.

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

4. Findings

In this section we describe the models developed at the Met Office and the processes by which they are developed.

4.1. The Met Office Unified Model

The Met Office maintains a common suite of Fortran routines (the *Unified Model (UM)*) for its Numerical Weather Prediction (NWP) and climate models. This code base has, arguably, been continually evolving for at least 30 years, certainly so since 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, which was used to provide data for the IPCC 2007 assessment [18], a newer generation of global environment models, HadGEM1 and HadGEM2 (the latter will be used in the next IPCC assessment), and a new research model, HadGEM3.

Most of the code for the UM was developed in-house at the Met Office, at a single location. However, as the range of expertise needed to develop climate models has grown, it has become increasingly hard to provide all the necessary expertise in-house. In the last few years, the Met Office has participated in a number of consortium efforts that complement its in-house expertise. These have led to the inclusion of UKCA, the UK atmospheric chemistry model, developed by a group of academic research labs, and NEMO, a state-of-the-art ocean model developed at the Centre National de la Recherche Scientifique (CNRS) in Paris.

The current release of the UM is about 830,000 lines of Fortran source code. The code was maintained using CVS for a long time, but two years ago the Met Office adopted a new code management system, FCM, based on the open source tools Subversion and Trac [13]. Currently they release 2-3 versions of the UM per year.

Figure 2 shows the growth in size of the UM over the last 15 years. Discontinuities in the steady growth of the model represent addition or replacement of major components: At version 5.0, the old dynamical core of the model was replaced entirely, removing much of the complexity of the

old scheme, whilst improving its functionality. UKCA was added at v6.2, and the old ocean model was replaced with NEMO at v7.0. Two releases represent ports of the model, with no new functionality: v6.0 was a port from the Cray to the new NEC supercomputers (which also coincided with the move of the Met Office headquarters from Bracknell to Exeter), and v6.3 was a port to FCM, which also included a cleanup of the file structure. Note the deliberately faster release cycle after the adoption of FCM.

Interestingly, the time taken to perform a climate run hasn't changed over the life of the UM, because climate scientists take advantage of increases in supercomputer power to increase the resolution and complexity of the models. 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 1-2 decades of simulation, which can still take a couple of weeks, depending on the model configuration. Although the older models can now be run on desktop machines, much of the leading edge science concerns 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 a User Interface for configuring model runs (the UMUI), 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.

4.2. Software Evolution

As Figure 2 showed, the UM has undergone steady evolution throughout its history. Drivers for change come from several directions (see Figure 3). First, advances in the underlying science lead to improvements in the way that physical processes are represented 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 data sets, and with runs of models from other centres, to identify areas of weakness. Fourth, operational concerns sometimes lead to changes, for example to allow the models to run in new hardware configurations. In the fall of 2008, the Met Office will acquire new IBM supercomputers, which will entail a large porting effort.

Occasionally, scientists make opportunistic changes to the UM, to improve speed or to tidy up the code base. These tend to be treated as lower priority, except in cases where such changes are likely to lead to major improvements to the operational forecasting models. The prevailing culture discourages such changes, in part because a scientist proposing such a change must demonstrate that it will have no negative effects on the accuracy and performance of any of the operational models.

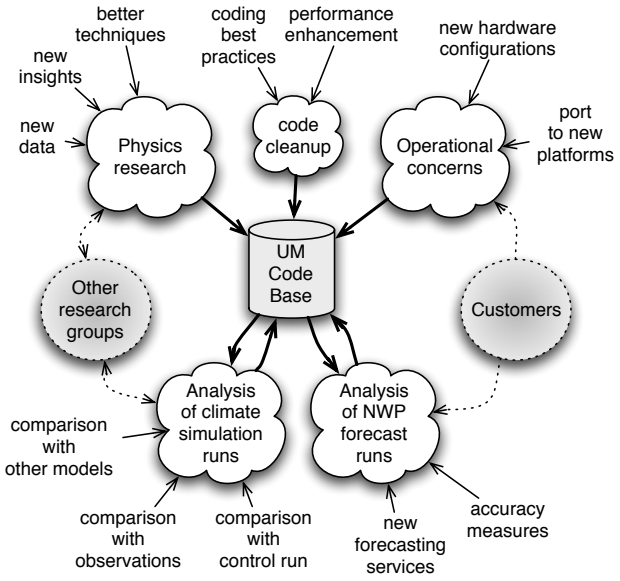

Figure 3. Sources of change to the UM code

These change drivers are largely internal to the Met Office. The Met Office's customers do not interact with the software directly, and so customer requirements affect the change process only indirectly, for example when forecasters identify a demand for new types of weather forecast data, or when the computational load needed to generate deliverables for sponsors restricts the availability of supercomputer nodes for development work. On the other hand other climate research centres do run the models themselves. Change requests from peer institutes and science review committees are filtered through Met Office contacts.

These change drivers lead to a number of requirements conflicts. For example, there is often a trade-off between improving the *skill* of the models in reproducing observed weather and climate variations, versus improving the *scientific validity* of the physics schemes. The models have to be tuned empirically by adjusting the parameterization schemes, because both the models and the parameterizations are only approximations of the real physical processes. Such tuning means that improvements that are more scientifically justifiable sometimes decrease the model's skill. The scientists sometimes talk of the models getting a good match with observations for the wrong reasons[1].

The conflict is complicated by the use of the UM for operational forecast models, which have strong constraints on performance (forecasts must be delivered on time) and accuracy (the Met Office has annual targets for improvements in weather forecast accuracy). Changes, such as scientific improvements, may individually have negative effects on performance or forecast accuracy, and so might be

[1] This problem resembles the overfitting sometimes observed in benchmarking efforts [23].

resisted by the user base unless part of a combined package of changes that has an overall positive effect. Such conflicts are sometimes dealt with by including several alternative versions of various physics schemes within the UM, which can be selected when different models are built.

4.3. The Development Process

The software development processes used for the UM have also undergone significant evolution over the life of the code. In particular, the adoption of FCM 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, organised like the 'onion' model often observed in open source projects. At the core, about twelve people from the two IT support teams (Met R&D and CR) control the acceptance of changes into the trunk of the UM. 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 sections of the UM (e.g. atmosphere, ocean, boundary layer, dynamical core, etc). 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 the local experts for knowledge about how to configure the model for particular runs, and keep track of the experiments performed with the model. About 100 Met Office scientists contribute the bulk of the code changes for any given release. These scientists regularly use the models in their scientific research, and make changes for any of the reasons given in Figure 3. Finally a broader group of scientists both within and outside the Met Office make occasional use of the models, and might suggest potential improvements.

Releases of the UM are planned on a regular schedule, typically every 4-5 months. All changes to the model are captured as tickets in Trac, and tickets are allocated to upcoming releases using an agile planning approach. Approximately three months into the release cycle, there is a deadline for tickets to be included for that release, and one month later there is a code freeze (the IT teams refer to it as *frosting* rather than a *freeze*, as some changes are still permitted). One further month is allowed for the IT teams to ensure all configurations of the models 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 trunk of the UM. First, a scientific review is done by the relevant code owner. Significant changes are typically discussed with code owners in advance, to explore the scientific justification and relative priority, while smaller changes are submitted for review once they are complete. Second, an IT team member performs a system review. This focuses on coding standards, basic code hygiene (e.g. 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 change into the trunk (no more than four per day), 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 a very lightweight project management. The management philosophy at the Met Office Hadley Centre is that, as a centre of excellence, the scientists should have plenty of freedom to set their own research goals. Each of the named climate models is defined as a project using the PRINCE project management method, to identify the strategic scientific objectives, allocate resources, and manage risk. But within a project, 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.

4.4. Verification and Validation

V&V processes are dominated by the understanding that the models are imperfect representations of very complex physical phenomena. Instead of reasoning about code "correctness", the scientists at the Met Office treat the models as evolving theories, and design experiments using the models to test specific hypotheses. They constantly make choices about model resolutions and timescales, and which modules should be coupled in and which should be parameterized.

Analysis of the model outputs is perhaps the most common activity, to look for causes of error and ways to reduce it. The scientists have developed a set of visualizations for displaying the results from model runs. Figure 4 is a typical example, showing model data plotted across a global map. Such visualizations appear everywhere: on people's desks, pinned to the walls, passed around in meetings, and on powerpoint slides used in scientific seminars.

During development, model runs are set up as experiments. 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. Automated tools provide the necessary visualizations for all the relevant variables in what is termed a *validation note*. For example, figure 4 shows the effect on mean pressure at sea level (PMSL) for December to February (djf)

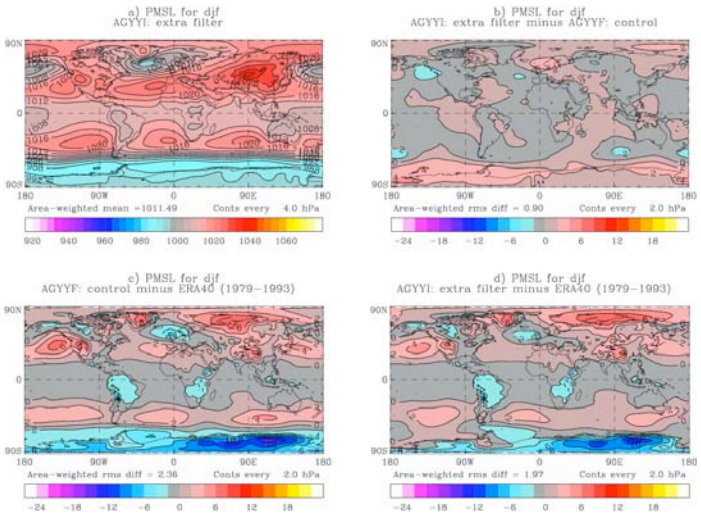

Figure 4. Example Validation Note

of the addition of a new polar filter. 4(a) shows the new model run, 4(b) shows the new run minus the control run, 4(c) shows the control run minus the observational data, and 4(d) shows the new run minus the observational data. These maps give instant visual feedback on the experimental results, in this case that the change has produce the expected improvement in the antarctic.

This approach means that the scientists are performing continuous integration testing, but don't view it as such, because for them it is part of the business of "doing science". They don't need "finished" software to perform these experiments – they need to 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 run-time switches) to ensure previous experiments can be reproduced. However, reproducibility can only be guaranteed if the outputs of the old run and the new run are exactly identical (down to the least significant bits). Also, the models are designed to be run on different platform configurations, for example over a single or multiple processors. Bit comparison tests can check that all configurations give identical results. Bit comparison tests are taken very seriously. The IT staff run these tests, and maintain a wiki page listing changes that break bit-level reproducibility.

This use of bit-level comparison has not been observed in other fields of scientific computation, and may be unique to the climate modeling community. A major advantage is that it can be used to shortcut (and partially automate) both regression and configuration testing. 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 (but not perfect) indicator of reproducibility over longer runs. The overnight test harness performs a number of such tests. The disadvantage is that it 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. The practice dates back to the very early days of the UM.

Another V&V strategy is to compare results with other models. Such comparison is formalized in a series of community-wide Model Intercomparison Projects (MIPs) and the use of *model ensembles* [5]:

- Multi-model ensembles are used to compare models developed at different labs on a common scenario.
- Multi-model ensembles using variants of a single model are used to compare different schemes for parts of the model, e.g. different radiation schemes.
- Perturbed physics ensembles are used to explore probabilities of different outcomes, in response to systematically varying physical parameters in a single model.
- Varied initial conditions within a single model are used to test the robustness of the model, and to better quantify probabilities for predicted climate change signals.

Model comparisons are also used informally, as a strategy for investigating sources of error. For example, a suspected problem in the soil hydrology module was investigated by comparing results with models from other labs. This led to the discovery that the coder had interpreted a formula from a published paper as a log_e, when it should have been log_{10}; other groups had interpreted it correctly.

Overall code quality is hard to assess. During development, problems that prevent the model running occur frequently, but are quickly fixed. Most of these are model configuration problems rather than defects in the code. Some types of error are accepted as modelling approximations, rather than defects. For example, errors that cause instabilities in the numerical routines, or model drift (e.g. over a long run, when conservation of physical properties such as mass is lost) can be complicated to remove, and so might be accommodated by making periodic corrections, rather than by fixing the underlying routines. The combination of continuous integration testing by the scientists, and bit reproducibility tests catch most errors prior to release.

We have not attempted a detailed analysis of post-release code defects. Over the last six releases, there were an average of 24 "bug fix" tickets per release, against an average of 50,000 SLOC touched per release, suggesting that 2 defects per 1,000 changed lines of code made it through the testing and review process for the previous release (and were subsequently discovered)[2]. However, these numbers must

[2]this suggests a defect density for the current release of 0.03 defects per KSLOC ($\approx$ 24 latent defects in 831,157 SLOC).

be interpreted carefully because many of these "bug fixes" represent defects that *were* detected, but treated as acceptable model imperfections in previous releases.

4.5. Maintaining a Shared Understanding

Scientists at the Met Office use a number of different strategies to maintain a shared understanding of the software. Formal design documentation exists for the UM, but is only sporadically updated. The scientists working on the model rely heavily on face-to-face communication, together with a large number of number of "informalisms" [21].

Face-to-face communication dominates. The large open plan office encourages this. It is possible to reach most of CR and Met R&D without traversing any doors or stairs. The office culture discourages noise, but many short 1-on-1 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, e.g. ongoing problems with the Indian monsoon.

The teams also make extensive use of electronic media for informal communication and coordination. A site-wide wiki is used extensively as a repository for informal documentation (design notes, to-do lists, task status reports, glossaries, etc). Site-specific newgroups are used for both social interaction and technical communication. For example, the IT team use them 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 the control structure of a new scheme. Written descriptions of designs, defects, and potential improvements all focus primarily on the underlying equations. Test results are described almost exclusively through the use of graphs of the results, focussing primarily on root mean squared (rms) error against observational data. Some scientists use the wiki as an electronic lab book, creating a page to describe each experiment, to summarize the setup and results, and link to the detailed validation notes.

When asked about the major challenges in their work, nearly everyone we interviewed described some aspect of 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).

Finally, we noticed some evidence of ontological drift [20]. For example a proposed glossary entry on "bit comparison" on an internal newsgroup revealed some people took it to mean the regression tests, while others took it to mean the platform configuration comparisons. Similarly, an open meeting to discuss a new initiative on seamless assessment revealed that several participants had very different understandings of key terms such as "traceability" [3]. Robinson and Bannon [20] suggest that such drift is normal in technical communities; our study indicated the scientists at the Met Office were effective at using the informal communication channels to identify and resolve such drift.

4.6. Community Models

The fact that model development has taken place at a single site appears to be important. Randall [17] reports that all existing GCMs were developed at large research labs, with no geographically distributed development, and suggests that the complexity of the coupling prevents it. 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 MOM ocean model developed at GFDL in Princeton, NJ. Its replacement, NEMO, was developed at CNRS in Paris. Such transplants allow a modeling group to tap into expertise available elsewhere, but is not without problems.

Technical challenges arise from getting a complex external component to work within the existing GCM architecture. The component may 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 may need to be modified. Seemingly trivial technical details (e.g. tiny differences in physical constants such as the radius of the earth) can cause problems, and are hard to track down. When the GFDL ocean model was incorporated into the UM, the modifications led to a fork from the GFDL model. Forking the model meant that although the Met Office gained a state-of-the art ocean model, it lost access to the ongoing scientific expertise at GFDL to keep the ocean model updated.

To avoid such forking with NEMO, a consortium agreement has been set up between the Met Office, CNRS, and other partners. Several coordination problems remain to be solved. NEMO is maintained by a small group in Paris, whose development cycle does not match that used at the Met Office. The consortium has to deal with tensions between the needs of individual members to customize and optimize NEMO for use in their own coupled models, versus maintaining the original design philosophy and flexibity of NEMO. The issues are similar to the problem of forkability in open source projects [7]. Although forking a project is always possible, it is invariably bad for every-

[3] The intended meaning is to do with the ability to compare the results of the same model when run at different resolutions and different timescales.

one involved to do so. Recognition of this fact strongly encourages open source communities to resolve such tensions democratically.

These external collaborations have become a key part of the Met Office's strategy in the last few years, to foster stronger collaborations with other research groups, and to tap into pools of expertise not available in-house. Examples include UKCA, an atmospheric chemistry module, JULES, a land surface scheme, and HiGEM, a community adaptation of HadGEM1 for higher resolutions. All of these are being developed by communities of academic partners in the UK. However, it is notable that in each of these community efforts, control of the core code base remains at a single site, with the partners tending to specialize in particular areas, and submitting their changes to the central site for inclusion in the reference model. These groups have not adopted the geographically distributed development used in open source communities. The result is that the core site can become a bottleneck.

Another tension in these community efforts comes from the different goals of academic partners, versus operational forecasting centres such as the Met Office. The former are geared primarily towards publication of scientific results. The latter have to satisfy hard goals on accuracy of their operational forecasting models, as well providing deliverables to customers for climate change consultancy. This has the potential to lead to a culture clash in the extent to which development cycles and review processes are formalized.

5. Discussion

Our study confirms many of the basic observations of the software practices of computational scientists reported elsewhere [4, 22, 2]. 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 (e.g. for code management and version control), they are readily adopted. The software has a very long lifetime, is written in an "old" programming language (Fortran), and performance issues are carefully balanced with maintainability and portability concerns. As in Kendall's study [11], we found that the developers had a strong shared scientific background, adopt an informal, collegial management style, with a culture of member participation and shared responsibility.

We found no evidence of Post's slowing down of "time-to-solution" [15] in this case study. The near-linear growth of the Unified Model over the past fifteen years indicates a steady growth of functionality, despite the growing complexity of the model. This steady growth is inconsistent with the findings of Lehman and colleagues, who found that for large commercial systems, the growth rate tails off as the software increases in size and complexity [12]. Instead, it more closely resembles the evolution patterns of open source projects reported by Godfrey [8]. We hypothesize that this is due to the many shared features with open source projects, which we discuss below.

A notable characteristic of this study is the broad set of approaches used for V&V. The use of bit-comparison tests as a technique for regression testing appears to be unique to this community, and 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 is part of "doing science". The extensive use of model inter-comparisons and model ensembles is also a distinct feature of this community.

The organisation behaves in some ways like an agile software development company, with all the developers housed in one very large open plan office, and a very strong reliance on informal communication channels. It uses many of the practices of agile development, including release planning, onsite customer, collective ownership, continuous integration, and risk management, but does not use any established agile process model. 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 agile practices also have this character: 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 is also consistent with studies of successful software startup companies [1], in which all the companies could be characterized as using a subset of "agile" practices, but each within a distinct, home-grown process model. Over time, it appears that these organizations evolve processes that are highly adapted to their "ecological niche". This probably only happens in stable, well-established teams with long history of working together, which was the case for the majority of companies examined in [1], and for the scientists working at the Met Office. Such observations also suggest that domain-independent process models (such as RUP, SCRUM, XP, etc) are simply irrelevant to these organisations.

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

- The release schedule is not 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.
- They control the code by having a small number of code owners and a much larger set of contributors, and a careful review process to decide which changes get accepted into the trunk.
- 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 developers all have "day jobs" - they're employed as scientists rather than coders, and only change the model when they need something fixed or enhanced. They do not to delegate code development tasks to others because they have the necessary technical skills, understand what needs doing, and because its 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 one anothers' contributions.

However, the Met Office lacks two characteristics that are usually thought of as key distinguishing traits of OSS projects: geographically distributed teams and a commitment to open source licencing. This suggests an interesting hypothesis: the success of open source projects may have more to do with a community of (very smart) 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.

6. 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 may 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 point of view, 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 is likely to be a difference in how the scientists perceive defects and bug fixes, compared to the use in the software engineering literature, because model imperfections are accepted as inevitable. Hence, any measure of defect density may not be comparable with that of other types of software.

The primary threat to internal validity comes from researcher bias. Data collection and analysis was performed by the first author, who has a software engineering background and is new to climate science. Analysis and followup discussions focussed on issues that seemed unusual or remarkable from a software engineering perspective. The second author provided a climate scientist's view on these issues as they were identified, and helped to correct any misunderstandings of the scientific practices.

We cannot claim that the observations in this case study generalize. As a climate modeling centre, 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.

7. Conclusions

Our goals in this study were to characterize the software development practices of climate scientists as a particular subtype of computational science. The study confirms many of the general observations of earlier studies. Notable differences include the sophistication and variety of the approaches used for V&V, and the absence of any obvious confirmation bias. Climate scientists accept that imperfections in their models are inevitable, and focus their attention on identifying and reducing modeling errors.

The Met Office has evolved a software development process that is highly adapted to its needs, which relies heavily on the deep domain knowledge of the scientists building the software, and is tightly integrated with their scientific research practices. The V&V practices are absorbed so thoroughly into the scientific research that the scientists don't regard them as V&V. However, they do appear to rely heavily on informal face-to-face communication to allow the scientists to develop a shared understanding of their models.

The software development practices at the Met Office share a number of features with both agile and open source development projects. 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 the deep expertise

and self-organisation of their developers. We hypothesized that under such circumstances, the developers will gradually evolve a set of processes that are highly customized to their context, irrespective of the advice of the software engineering literature. However, further research into such comparisons is needed to investigate these observations.

Post & Votta argue that computational science needs a new paradigm to address the prediction challenge [16]. They point out that most fields of computational science lack a mature, systematic software validation process that would give confidence in predictions made from computational models. Our study indicates that for climate science, at least as practiced at the Met Office, such model validation is routinely performed, as it is built into a systematic integration and regression testing process, with each model run set up as a controlled experiment. Furthermore, climate science as a field has invested in extensive systems for collection and calibration of observational data to be used as a basis for this validation. Climate models have a sound physical basis and mature, domain-specific software development processes. Hence it is hard to identify potential for radical improvements in the efficiency of what is a "grand challenge" science and software engineering problem.

Acknowledgements*: 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. Fig 3 is due to Damian Wilson. Primary funding for the study was provided by NSERC. Tim Johns was supported by the Defra & MoD Integrated Climate Programme - (Defra) GA01101, (MoD) CBC/2B/0417_Annex C5*

References

[1] J. Aranda, S. Easterbrook, and G. Wilson. Requirements in the wild: How small companies do it. In *IEEE International Requirements Eng. Conf. (RE'07)*, pages 39–48, 2007.

[2] V. Basili, J. Carver, D. Cruzes, L. Hochstein, J. Hollingsworth, F. Shull, and M. Zelkowitz. Understanding the High-Performance-Computing Community: A Software Engineer's Perspective. *IEEE Software*, 25(4):29–36, 2008.

[3] A. Capiluppi, J. Fernandez-Ramil, J. Higman, H. Sharp, and N. Smith. An Empirical Study of the Evolution of an Agile-Developed Software System. In *29th Int. Conf. on Software Engineering*, pages 511–518, 2007.

[4] J. Carver, R. Kendall, S. Squires, and D. Post. Software Development Environments for Scientific and Engineering Software: A Series of Case Studies. In *29th Int. Conf. on Software Engineering (ICSE'07)*, pages 550–559, 2007.

[5] M. Collins. Ensembles and probabilities: a new era in the prediction of climate change. *Philosophical Trans. of the Royal Society*, 365(1857):1957–1970, 2007.

[6] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/Libre Open Source Software: What we know and what we do not know. Submitted for publication, 2008.

[7] K. Fogel. *Producing Open Source Software: How to run a successful free software project*. O'Reilly, 2006.

[8] M. Godfrey and Q. Tu. Evolution in open source software: A case study. In *IEEE Int. Conf. on Software Maintenance (ICSM'00)*, pages 131–142, 2000.

[9] J. Hansen, R. Ruedy, A. Lacis, M. Sato, L. Nazarenko, N. Tausnev, I. Tegen, and D. Koch. Climate Modeling in the Global Warming Debate. In D. Randall, editor, *General Circulation Model Development: Past, Present, and Future*. Academic Press, 2000.

[10] R. Katz. Techniques for estimating uncertainty in climate change scenarios and impact studies. *Climate Research*, 20(2):167–185, 2002.

[11] R. Kendall, J. Carver, D. Fisher, D. Henderson, A. Mark, D. Post, C. Rhoades Jr, and S. Squires. Development of a Weather Forecasting Code: A Case Study. *IEEE Software*, 25(4):59–65, 2008.

[12] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski. Metrics and laws of software evolution-the nineties view. In *IEEE Int. Software Metrics Symp. (METRICS'1997)*, 1997.

[13] D. Matthews, G. V. Wilson, and S. M. Easterbrook. Configuration Management for Large-Scale Scientific Computing at the UK Met Office. *Computers in Science and Engineering*, 2008. In press.

[14] N. Oreskes, K. Shrader-Frechette, and K. Belitz. Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. *Science*, 263(5147):641, 1994.

[15] D. Post. The Coming Crisis in Computational Science. In *IEEE Int. Conf. on High Performance Computer Architecture: Workshop on Productivity and Performance in High-End Computing*, 2004.

[16] D. Post and L. Votta. Computational Science Demands a New Paradigm. *Physics Today*, 58(1):35, 2005.

[17] D. Randall. A University Perspective on Global Climate Modeling. *Bulletin of the American Meteorological Society*, 77(11):2685–2690, 1996.

[18] D. A. Randall, *et. al.* Climate Models and Their Evaluation. In S. Solomon, *et. al.*, editor, *Climate Change 2007: The Physical Science Basis. Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge U. Press, 2007.

[19] D. Rind. Complexity and Climate. *Science*, 284(5411):105, 1999.

[20] M. Robinson and L. Bannon. Questioning representations. In *2nd European Conf. on Computer-Supported Cooperative Work*, pages 219–233, 1991.

[21] W. Scacchi. Free/open source software development: recent research results and emerging opportunities. In *IEEE Int. Symp. on the Foundations of Software Engineering (ESEC/FSE'07)*, pages 459–468, 2007.

[22] J. Segal and C. Morris. Developing Scientific Software. *IEEE Software*, 25(4):18–20, 2008.

[23] S. Sim, S. Easterbrook, and R. Holt. Using benchmarking to advance research: a challenge to software engineering. In *25th IEEE Int. Conf. on Software Engineering (ICSE'03)*, pages 74–83, 2003.

[24] N. Stehr. Models as Focusing Tools: Linking Nature and the Social World. In H. von Storch and G. Flöser, editors, *Models in Environmental Research*. Springer, Berlin; New York, 2001.

[25] J. Sterman. The Meaning of Models. *Science*, 264(5157):329–330, 1994.

[26] G. Wilson. Where's the real bottleneck in scientific computing? *Am Sci*, 94(5), 2005.