Do Over or Make Do?

Climate Models as a Software Development Challenge

Steve Easterbrook

**Dept of Computer Science,
University of Toronto**

http://www.easterbrook.ca/steve

1

My background is in software engineering, having spent the last 20 years investigating software processes in aerospace, commercial, and scientific software development. I'm especially interested in issues of team coordination and communication, and how these impact software quality.

I've spent much of the past year completing a comparative study of software development practices at four major modeling centres. My study at the UKMO was conducted in 2008, and I published a preliminary paper describing my observations in the journal Computing in Science and Engineering. I visited the other three centres for a month each this year, and am still analyzing the data I collected, so most of what I say today should be taken as very preliminary results. My research methods are akin to anthropological studies: I examine the culture, practices, and tools of modeling teams from the point of view of an outsider, trying to make sense of why things are done in a particular way. Which means I'm especially interested in ways of comparing and contrasting climate modeling with other types of software engineering that I've studied.

Overview

Scientific Quality ≠ Software Quality

In the current generation of Earth System Models, scientific quality has always trumped software quality

…largely because of how they're built

Does this matter?

Yes: It hampers ability to deal with scale and complexity

Choices:

Make Do - work with existing models, iteratively refine them

Do Over - rebuild the models from the ground up

or…?

(Some insights from Agile Software Practices)

Conclusion:

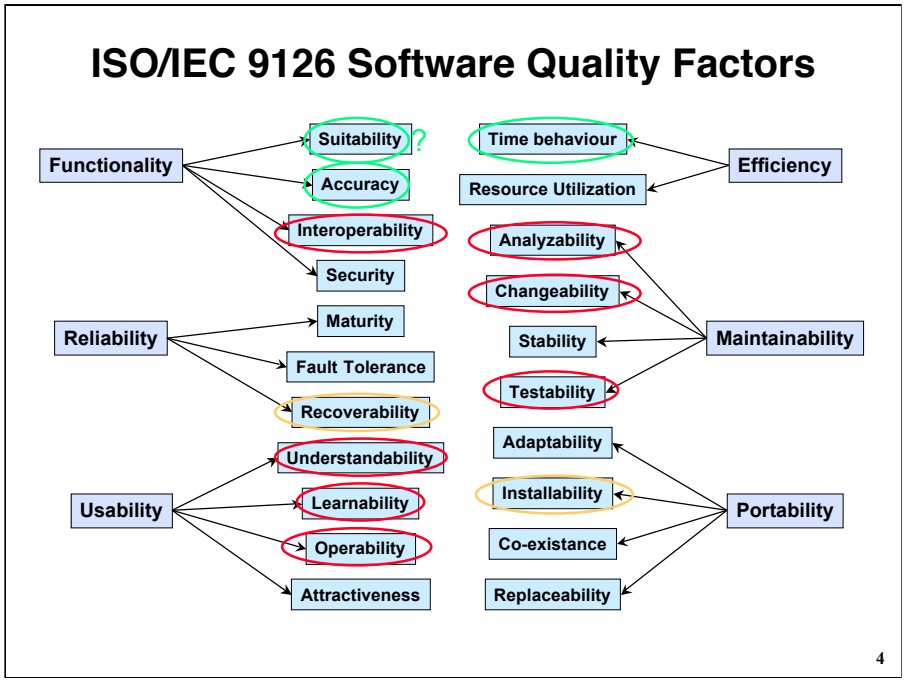
There are few (if any) ready-made solutions from software engineering

Instead, the community will have to build its own reflective praxis

3

Here's what I want to say this afternoon. I want to make the point that climate models are primarily judged on scientific quality rather than software quality, and that this emphasis is entrenched in the exploratory, iterative process by which they are developed (over a very long period). This produces models that are getting steadily better from a scientific point of view, but steadily worse from a software quality point of view. This might not matter, except that the models have grown dramatically in size and complexity over the past decade. Software qualities that have been neglected (e.g. understandability, usability, portability), now hamper the ability to develop the models further.

So, we could pose the question: is it time to re-build the models from the ground up using stronger software engineering principles? I'll try and answer this question with a quick look at issues of modularity and coupling in the current generation of earth system models. I'll argue that a "do over" is too risky, too expensive, and probably infeasible due to a lack of available expertise. So if we're stuck with the "make do" approach, what can we do? I'll argue that we can learn a lot from agile software development practices, which emphasize people and tools rather than processes and documentation. But I conclude that the challenge of developing climate models has a number of unique constraints that means other software engineering practices don't readily apply. So the community will have to develop it's own set of practices, building on what has been done already. And iterative improvement is not only okay, it's probably the only thing we can do.



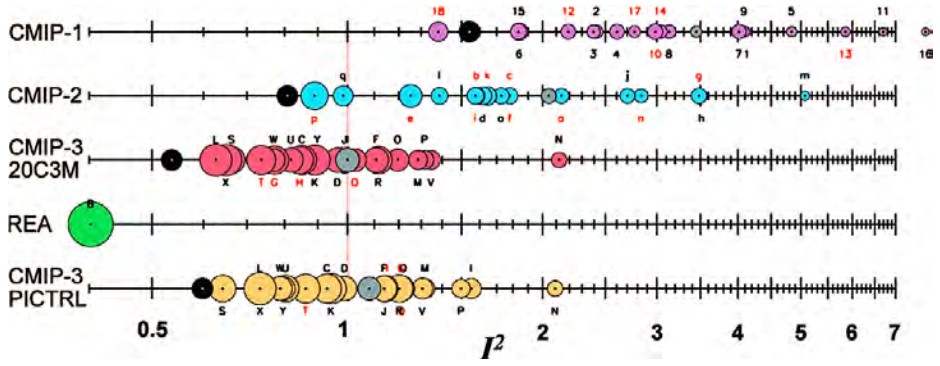
First, that point about software quality. Although we can broadly categorize quality as “fitness for purpose”, it’s useful to break it down into a number of factors, each of which might matter more or less for different kinds of software.

For climate models, I would argue that suitability (for the scientific questions being asked) and accuracy (in terms of model skill) have been the driving issues, and of course, processing speed (efficiency) has been vital.

Portability (especially installability) has been important at some centres, particularly those which do not have their own dedicated supercomputing facilities, and hence have been forced to make the model run on different platforms.

But the models are doing badly on most aspects of usability and maintainability - they’re not designed to be particularly usable or maintainable, because there’s only a limited pool of people available to do model development, and they’re too busy keeping up with the science to worry about designing for maintainability or designing for usability.

Model Improvements



- average I^2 across the model group

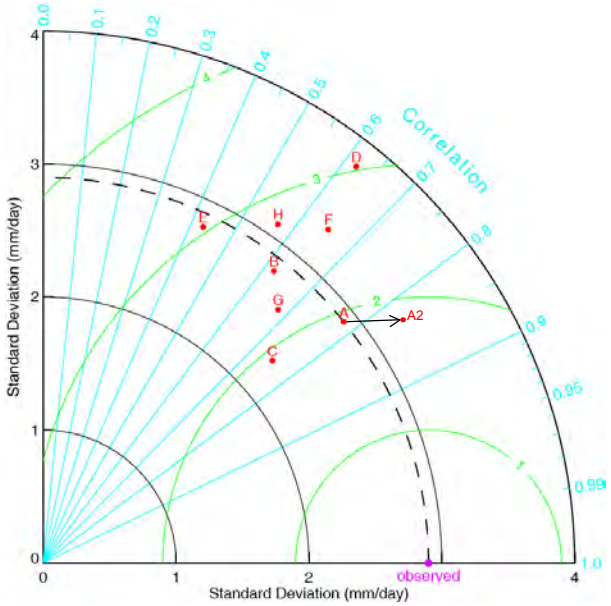
- I^2 of multi-model mean

Source: Reichler and Kim, BAMS 2008; 89: 303-311

5

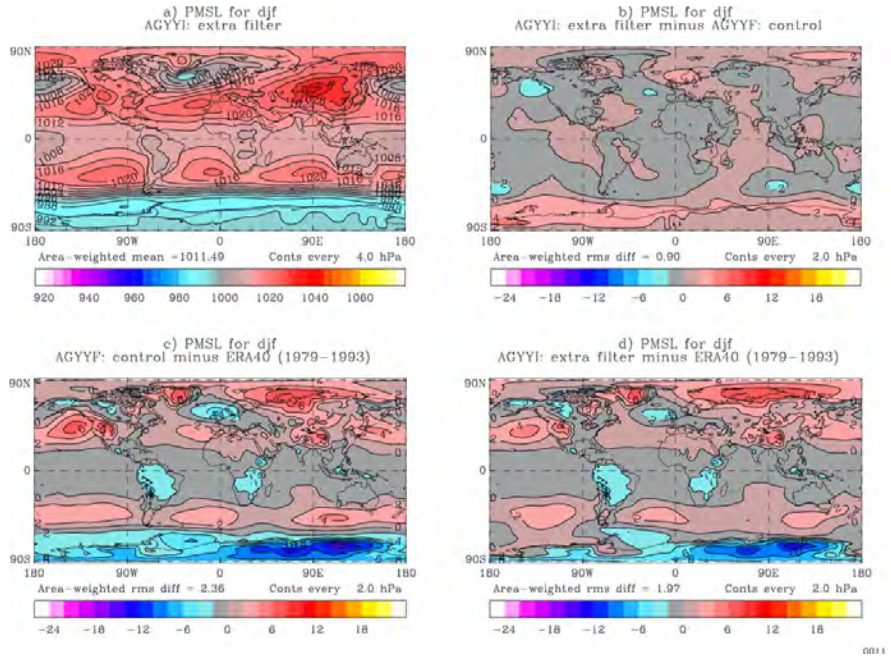
The emphasis on getting the science right first is clearly paying off - e.g. this analysis from Reichler and Kim shows a steady improvement in model skill over the various model generations. Here, model skill Is measured as an aggregate score in accurately reproducing observed climate variables.

Taylor Diagrams



6

And the community has evolved some sophisticated ways of visualizing how well the models reproduce both the mean and the variability seen in observational data.



And the focus on model skill is embedded in the everyday model development practices. A typical approach is to make a small, experimental change to the model which is expected to produce some improvement in skill in some specific earth system processes. The change is then compared to the old version, using observational data to assess whether the expected improvement was achieved.

Some changes are aimed not at improving skill, but at improving the theoretical basis for the model (sometimes at the cost of skill); but these changes are still assessed in the same way - the overall impact on model skill is carefully assessed and compared to expectations.

Key Success Factors

Highly tailored software development process
(software development is "doing science")

Basic practices: version control, automated test, …

Software developers are domain experts

Staff continuity + "ask the expert"

Shared ownership and commitment to quality

Unconstrained (but regular) Release Schedule

Benchmarking (e.g MIPS & ensembles)

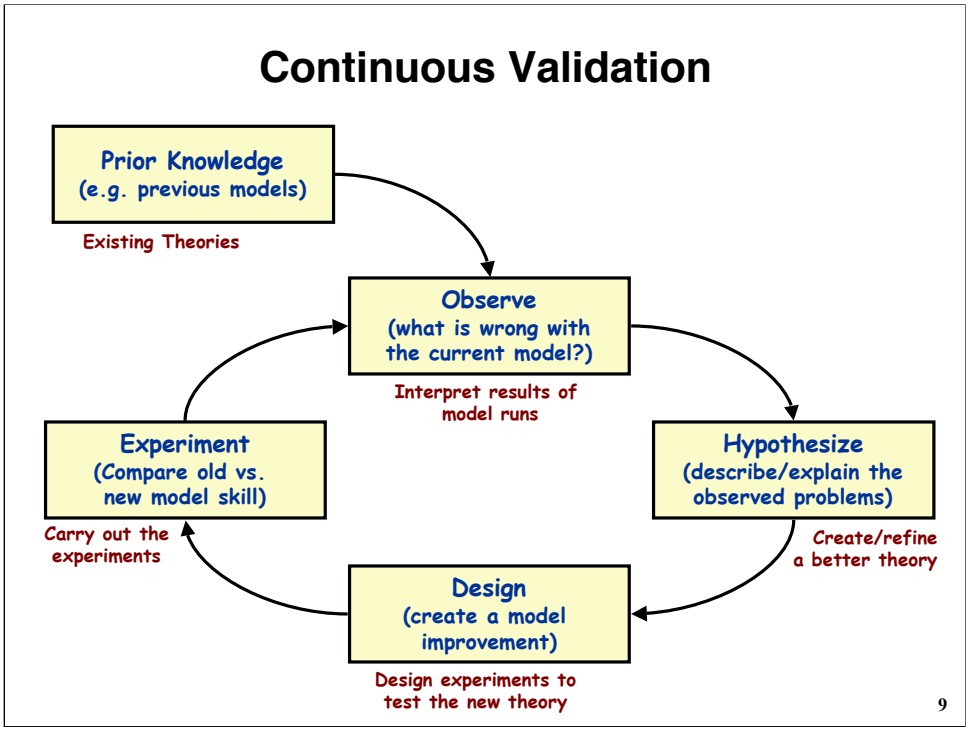
Openness ("Many eyes" validation)

Source: Easterbrook & Johns "Engineering the Software for Understanding Climate Change." Computing in Science and Engineering. 2009;11:65-74. 8

So, in my preliminary studies identified a number of factors that drive the success in creating the current generation of models and ensuring they are scientifically valid, with steadily improving skill.

Most importantly, the people developing the software are the experts - the scientists themselves - and there's good continuity (most of the modelers work with the same model for many years) and recognition of expertise (if you want to know something about a specific part of the model, you go talk to the expert for that part).

It also helps that the models are developed primarily for internal use - scientists are both developers and users - so there's no problems of misunderstanding the expectation of customers, nor any pressure from customers to release new versions on some predefined schedule.

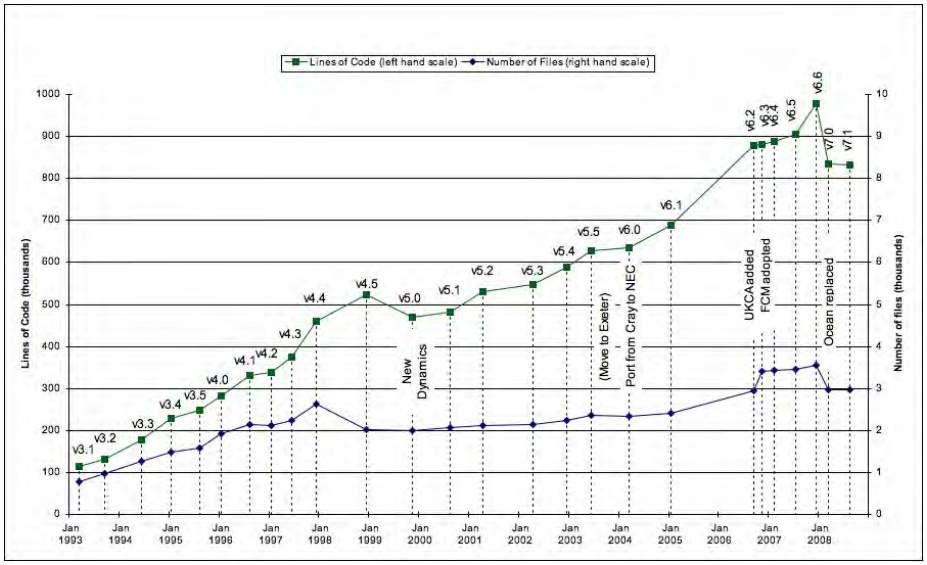


In effect, this means the model development process matches the process of scientific discovery. Each small change to the model represents a scientific hypothesis, driven by an observation that there's an area of weakness in the model, and a hypothesis about how to improve it; this is then followed by experiments to check that the hypothesis was correct, that the change did give the expected improvement (and didn't break anything else). And the cycle applies to remarkably small changes - often just a few lines of code. Which makes model development slow and painstaking compared to commercial software development (especially when you factor in the days of weeks of waiting for runs to complete). But it means there's a strong culture of what I might term "continuous validation" amongst the modeling community, and this experimentation process means that the the (core) community come to understand the models in great detail.

(A downside is that people outside the core modeling team don't get this understanding, and for them, them model becomes more opaque as it develops).

So this strategy has been very successful over the past couple of decades. But most labs now recognize they're facing new challenges as the models grow in size and complexity. Will the current approach scale to more complex earth system models?

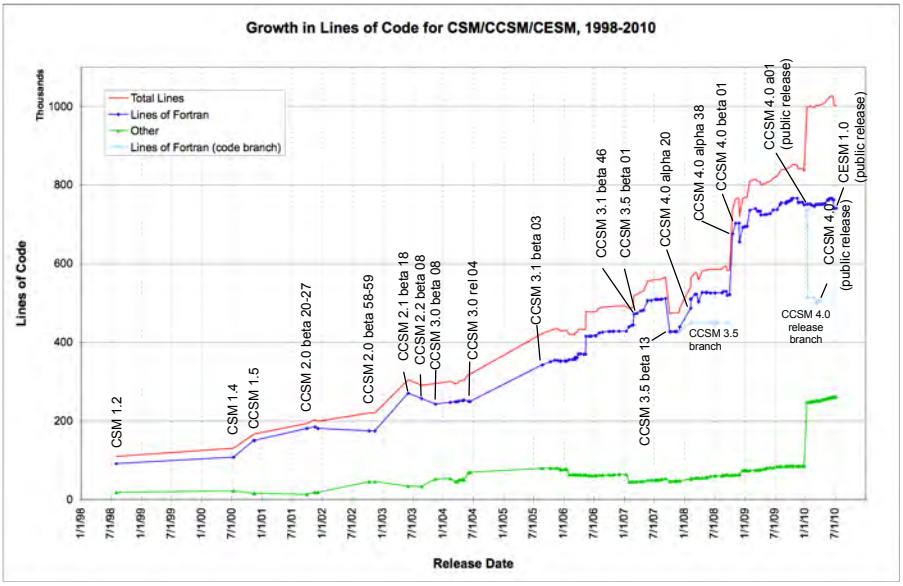
UKMO Unified Model Code Growth



11

Here are some indicators of that growth. This chart shows the growth of the UKMO Unified Model over the last fifteen years in terms of lines of code. The green line at the top is lines of code, while the blue line is number of files (roughly, Fortran Modules). Over a fifteen year period, the code base grew from 100,000 lines of code, to almost a million - an order of magnitude change.

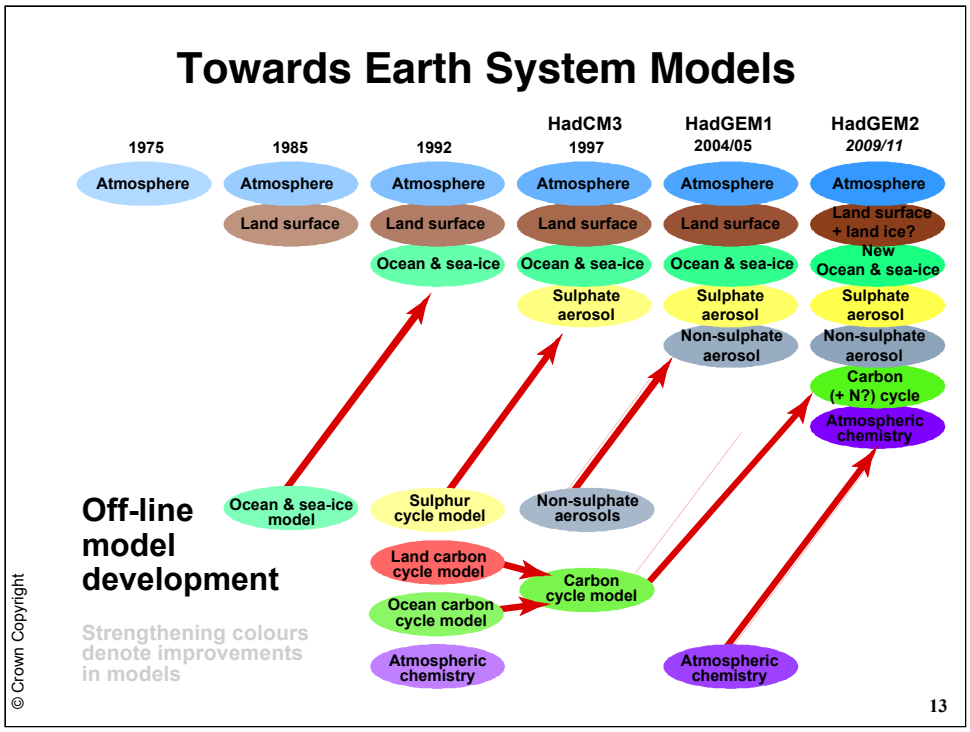
NCAR CESM Code Growth



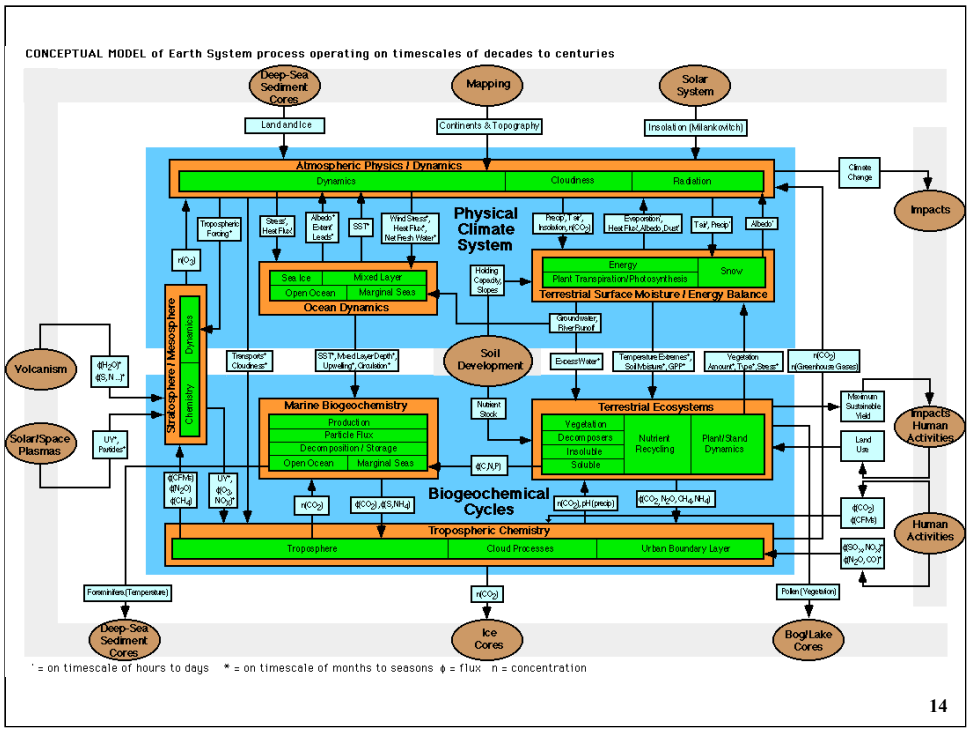
12

And the UKMO model isn't unusual. Here's a similar curve from NCAR's CESM, over twelve years, showing a similar order of magnitude growth.

[Note: the big jump in "other" in January 2010 was the inclusion of the model documentation into the code repository (previously the documentation was managed separately)]

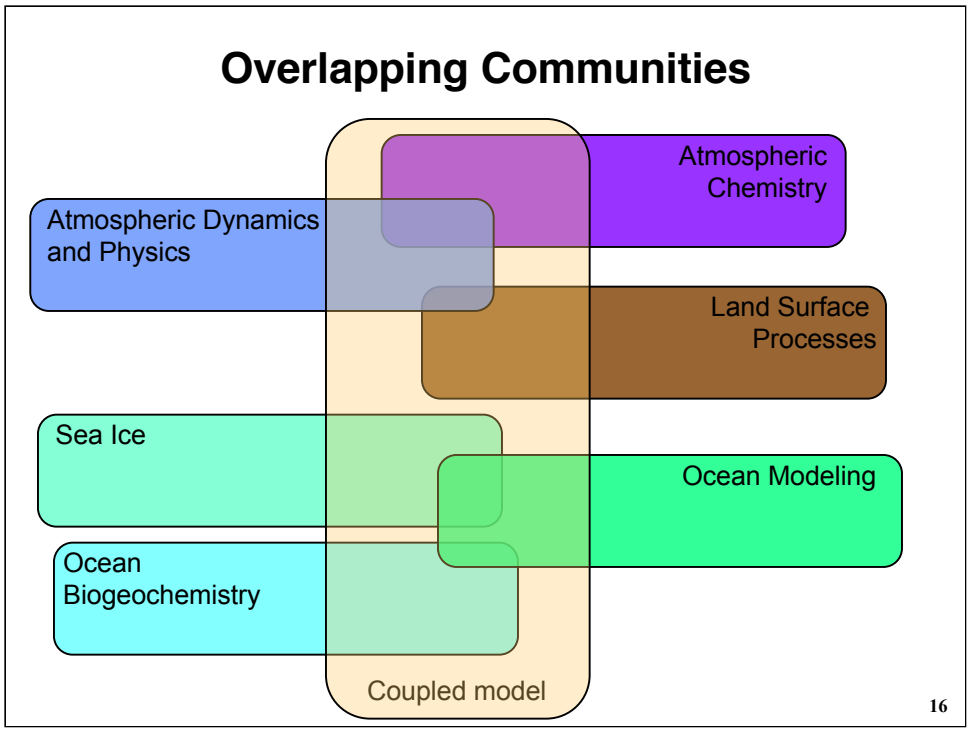


And one of the main reasons for this growth is a move from Atmosphere-Ocean Models to Earth System Models. Versions of this diagram are common in the modeling community, showing how the model now comprises a large number of different components representing different earth subsystems.



And the closest I can find to an “architecture” or “wiring diagram” for an earth system model is the Bretherton diagram. This shows the major components as orange boxes, with data flows between them shown as arrows. The various models differ slightly in exactly how these major subsystems are sliced up into separate components, and the nature of the coupling software that glues them back together.

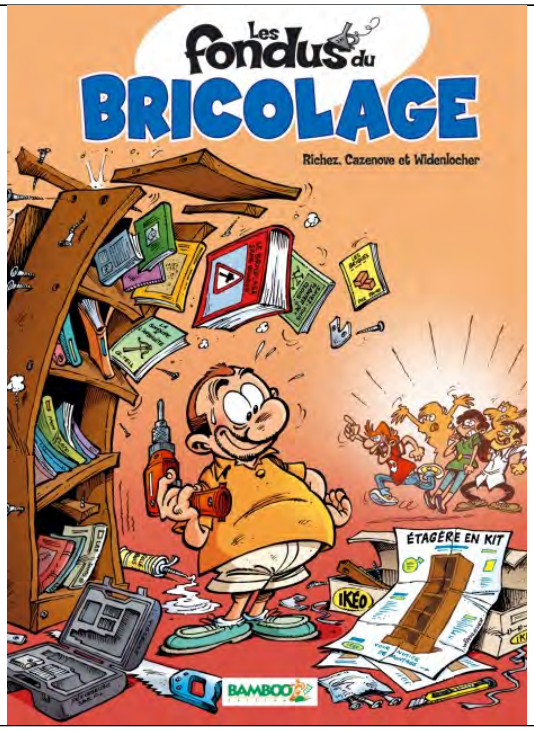
But these diagrams are too neat. It doesn't really come together like this.



In reality we have a number of different communities (with some overlaps), each building their own models of specific earth subsystems, typically for their own use as stand-alone scientific tools. A coupled model is then a complex negotiation between the needs of these individual communities and the kinds of component needed to construct a coupled earth system model. These various communities keep evolving their own models, often for their own purposes, so maintaining a coupled model is an ongoing challenge.

Note that I didn't draw this diagram to scale. If the coloured shapes represent the size of the community building the models, then the coupled model community should be tiny relative to the various specialized communities. A major problem is that very few scientists have the skills and motivation to develop and analyze coupled models (as opposed to the individual components from which they are constructed).

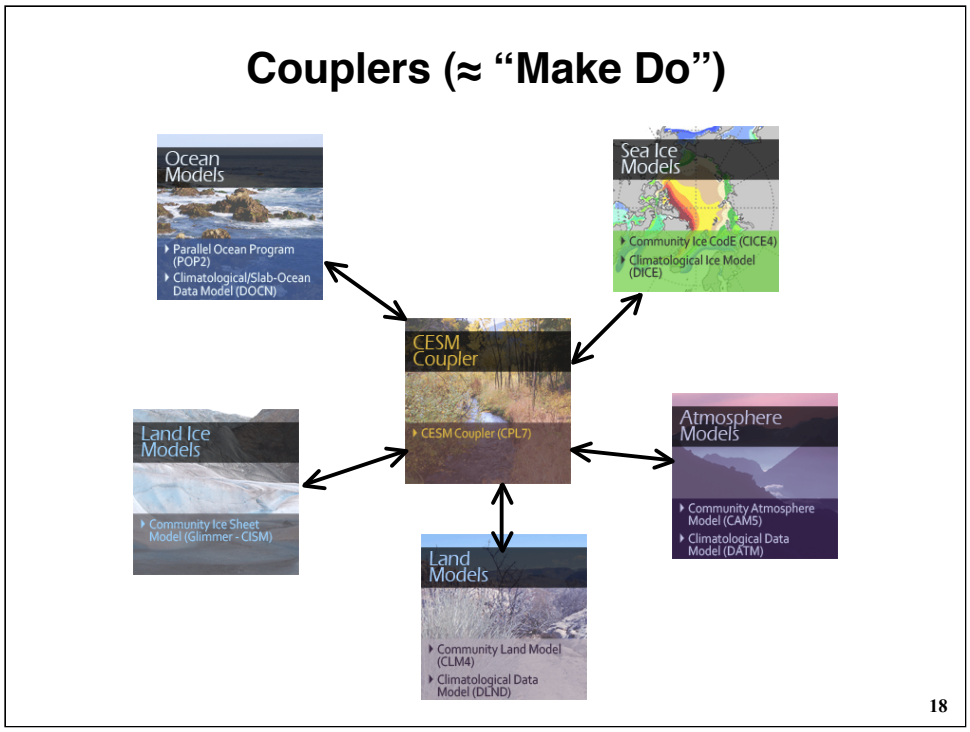
And yet, if the diagram represents the demand from policymakers and the public for information (e.g. the IPCC process), then the coupled model blob would be much *bigger* than the other blobs. So we have a serious problem: too few modelers focus on coupled earth system models compared to the demand for uses of these models.



17

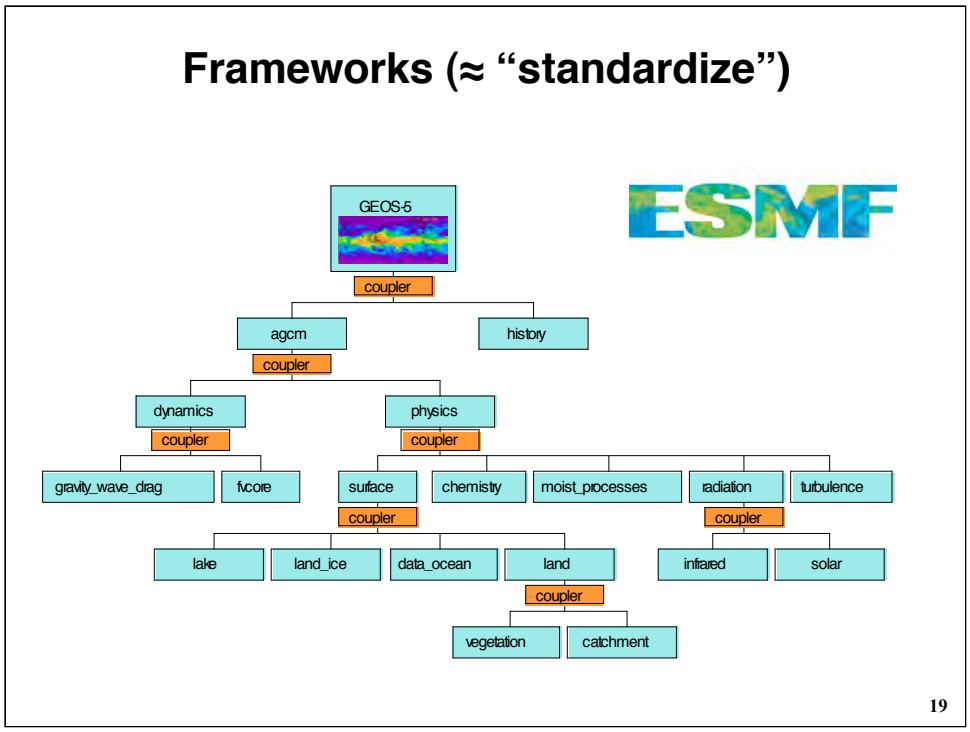
Which means that often, the job of putting together the components to build a coupled model involves a lot of software hacks - the components weren't designed to be put together in the coupled system, so the coupled model is never as elegant (in software terms) as the diagrams indicate.

My term for this is "scientific bricolage" - you build the coupled model out of whatever pieces are available, and make do with what you find.



So, a dominant approach is to build ever more complex couplers, which do the job of gluing together the component models, handling re-gridding, timing issues, data transformation, etc.

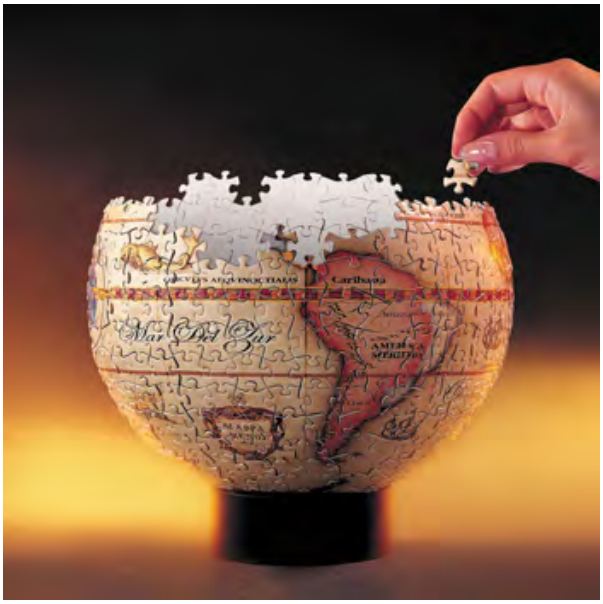
And a growing set of scripts that handle all the idiosyncrasies of the individual components, as they each have their own needs for Building, Configuring and Running them.



A different approach is represented by Frameworks, perhaps best represented by ESMF, the Earth System Modeling Framework. This isn’t so much a “make do” approach, but it’s not a “do over” approach either. It’s somewhere in between, perhaps best thought of as an attempt to create and promote standardizations, so that model components can be plugged into the framework, not necessarily as monolithic, idiosyncratic modules, but as a set of code routines that can be composed in systematic ways to build a coupled model system. But like all standardization efforts, acceptance by all the various modeling communities can be slow, in part because it requires each sub-community to change their codes in ways that sometimes get in the way of their own internal goals, and for which most of the benefits are for the broader community. And where the framework provides wrappers rather than standardized interfaces, it simply hides the coupling complexity in the wrappers, making the coupled model even harder to understand.

ESMF has been very successful in bringing the US modeling community together, and creating interchangeability between model components, but hasn’t improved the software quality of the various model components.

Complete Re-build (= “Do Over”)



20

And there don’t seem to be any attempts at a “Do Over” anywhere in the community, at least in the last decade. Where there are attempts to build new models (typically individual components, rather than coupled systems) they tend to be inspired by a different approach to the numerics, or a different grid system, rather than an attempt to re-build for software quality reasons.

Why no "Do Over"?

Cost

- 50 scientists x 20 years = 1,000 person years ≈ $150 million
- + supercomputing facilities for dev and test ≈ $200 million

Community

- Too few scientists with aptitude and motivation for earth system modeling
- Too few software engineers with necessary science background

Forking

- New model splits the community
- Lose access to the latest science

21

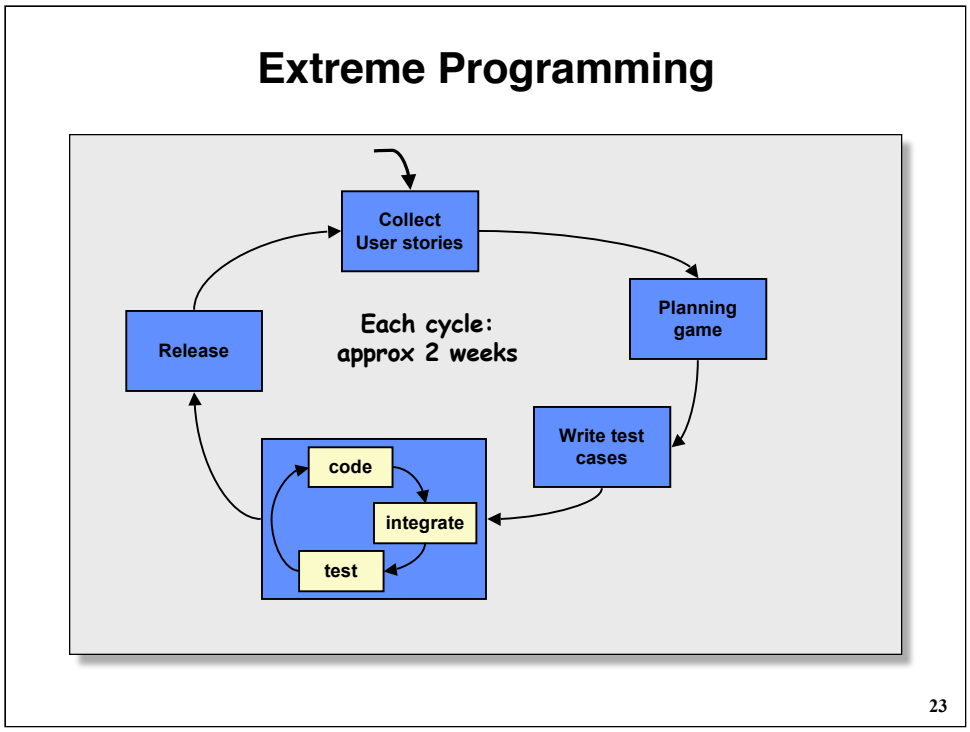
Actually, I don't think a "Do Over" is a viable approach, for a number of reasons. One is the cost. My back-of-the-envelope calculation of the cumulative cost of a current fully-coupled earth system model is around $350 million - representing the work of around 50 scientists working over a 20 year period. A rebuild might be done with a much smaller team over just a few years, but will still require substantial computing facilities - still in the order of hundred million dollars.

And even if we were willing to allocate funds to this (rather than using them for new science), we would have great difficulty finding the people to do this. The modeling community is already too small, short of scientists who understand coupled models, and short of software engineers who have enough science background to contribute.

And there's another downside - a rebuild would take several years, during which time the science will continue to move on. We're then effectively "forking" the community, so that we have distinct "science" and "engineering" models, making it increasingly harder to get new science advances into the engineering models.

The experience is likely to be similar to that of Netscape, who spent several years rebuilding their web browser from the ground up, only to discover that nobody wanted it because by the time it was ready, browser technologies had moved on (see http://www.joelonsoftware.com/articles/fog0000000069.html)

So, what can we do?



I think a lot can be learned by comparing model development processes with the agile software development philosophy. For those not familiar with this, “agile” represents a revolution in thinking about how software should be developed. It started in the 1990’s, largely in response to what some software developers perceived to be increasing bureaucracy of software development processes that deprecated the skills and expertise of individual software developers, and imposed a process engineering approach that stifles innovation. They argued that this removes the responsibility for quality from the software developers themselves, and places that responsibility in organizational procedures, which is a recipe for disaster.

Part of the agile philosophy is that you can’t successfully specify what a customer wants up front - you have to discover it by building a little, getting the users to try it, and using that experience to decide what to build next. One version of this, known as extreme programming, suggests this should be done in 2-week iterations. So you only plan to build what can be successfully implemented and tested within the two week cycle, and don’t worry about longer term planning (because longer term plans will always turn out to be wrong).

Notice the similarity with how I presented the model development cycle. Two weeks might be a little too short, but the essential iterative, exploratory process is there.

Agile software developers use a large number of tools and techniques to keep things on track, so we can examine these for ideas. But there’s a problem - agile development has been remarkably successful for

Use of Agile practices:

- ✓Collective Ownership
- ✓Configuration Management
- ✓Continuous Integration
- ✓Feature-driven development
- ~ Frequent small releases
- ✓Onsite customer
- ~ Organization-wide process
- ~ Organizational training
- ✗ Pair programming
- ✗ Planning game
- ✓Peer reviews
- ~ Process & product quality assurance
- ✓Project monitoring & control
- ✓Project planning
- ✗ Refactoring
- ✗ Requirements management
- ~ Retrospective
- ✓Risk Management
- ✓Simple design
- ✓Tacit knowledge
- ✗ Test-driven development

24

Many of the techniques used by Agile software companies are already widely in use by earth system modellers (although rarely given the same labels).

But what's noticeable is what isn't used, and here I think there's some scope for trying out these ideas withing the modeling community:

- refactoring, which continually re-structures the code to improve modularity and maintainability. In most cases, this is not used in earth system modeling because it breaks reproducibility. If you re-structure the code, the compiler will optimize it differently, which means you lose the ability to exactly reproduce older runs.

- test-driven development, in which test cases are written before the code is written, and are maintained as part of the code base, along side the code.

- pair programming, in which two developers work together on code changes, sitting next to each other and providing instant peer-review as the code is written.

But note that no agile software organisation uses all of these practices anyway - each organization figures out for itself which practices work for them, and which do not. So there's no package deal here, just lots of ideas that seem to have worked in some settings.

ESMs are not like other software…

- Don't know what to build in advance
- Models are always imperfect
- Cannot distinguish defects from approximations
- Many ways to handle imperfections
- No oracle for correctness
- Defects are not mission-critical
- Model surprises are useful
- Fidelity to real world physics not always needed
- Noisy real world data
- Model never finished (all runs experimental)
- Huge societal importance
- Politically motivated critics
- Developers are domain experts
- Developers are end-users
- Coding inseparable from 'doing science'
- Very smart people
- 25+ competing models
- Openness & shared ownership
- Unconstrained release schedule
- Stable top-level architecture
- Product line (many configurations)
- Submodels have independent lives
- Preference for Fortran
- Emphasis on integration testing
- Few resources for software infrastructure

25

And it's important to understand some of the unusual or unique constraints for the earth system modeling community, which makes this type of software development distinct from other types of software, especially commercial practices.

I've talked about some of these, particularly, the inseparability of coding from doing science, and the fact that the coupled models are constructed from components that have independent uses and are continually evolving.

The slide lists all the other dimensions I've identified, along which earth system modeling *might* differ from other forms of software engineering.

Conclusions

Appropriate people + Appropriate tools

Community building + Sharing of good practices

26

So my conclusion is that improving software quality and addressing the scaleability challenges will depend more on the expertise of the modeling community and the use of appropriate tools, than any particular software development processes.

The community of developers won't necessarily look like like commercial software developers, and the tools they use won't necessarily look like commercial software development toolkits, because this type of software has many unusual characteristics.

We can learn from agile development practices (because they match existing model development techniques in many ways), but what really matters is the development of more of a community approach that reflects more and shares information about what software techniques they are using and how well they are working.

And growth of this community: we need more people who have both good software development skills and sufficient geosciences background to work on coupled earth system models. The community is way too small at present.