

Software Teams as Knowledge Communities

Steve Easterbrook

Email: sme@cs.toronto.edu
Blog: www.easterbrook.ca/steve
Twitter: @SMEasterbrook

Except where otherwise noted, this work is licensed under
http://creativecommons.org/licenses/by-nc-sa/3.0/

For this talk, please:

Copy, share, adapt and remix;

Photograph, film, broadcast;

Blog, live-blog, tweet, post video of;

Provided that:

You attribute the work to its author and respect the rights and licenses associated with its components.

Note: This slide is adapted from one by @ethanwhite

Key Idea

We focus too much on studying software processes and tools…

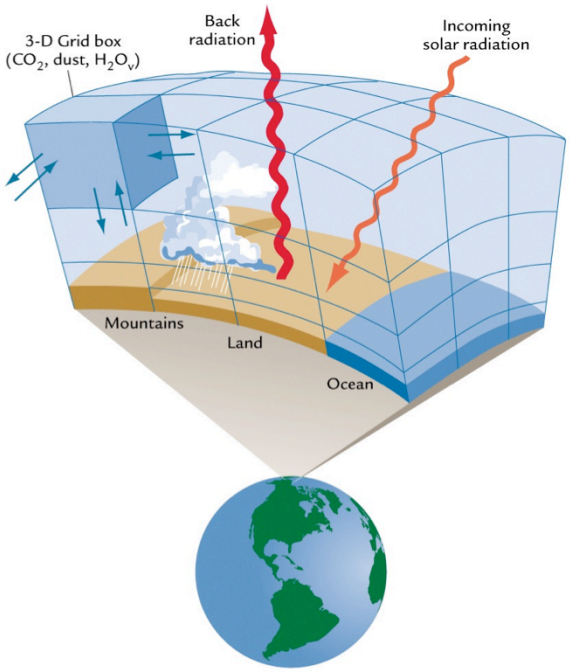
…and not enough on *what software teams know*, and how they come to know it

3



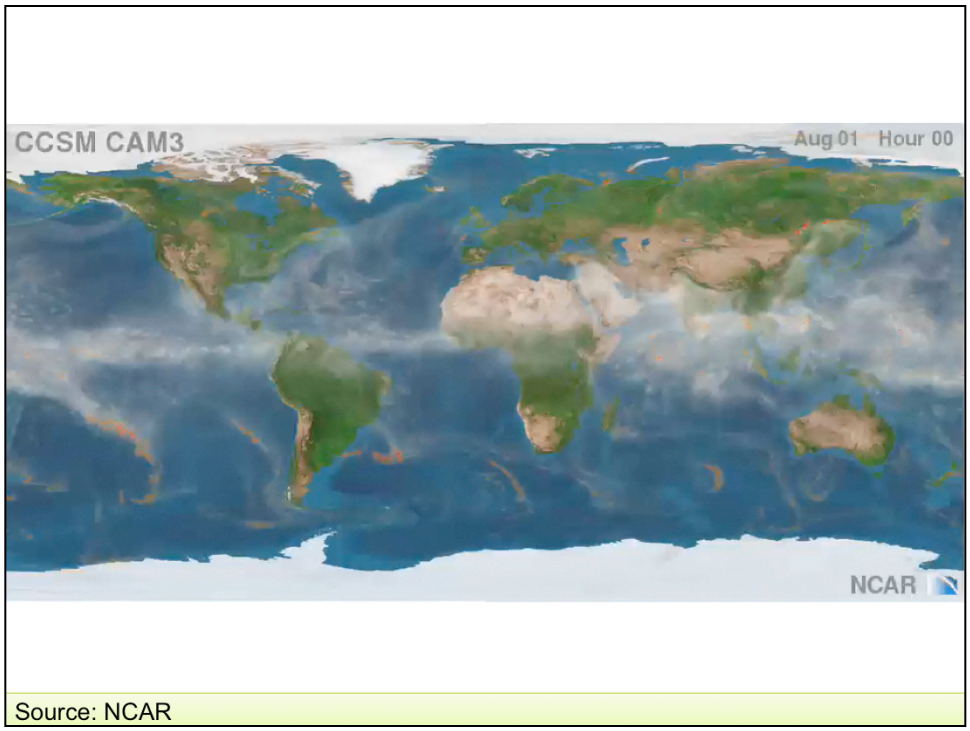
Most of the observations in this talk come from a series of ethnographic studies of how climate scientists build climate models. I started these studies expecting to be able to help the climate modelers with better software engineering techniques. While some of the software engineering is very ad hoc, for the bigger models, the labs have developed a very disciplined software development process. There are some opportunities to adopt new tools and techniques, but many of the obvious candidate tools don't readily apply.

They use a highly tailored software development process that is highly embedded in their scientific methodology, and offers some interesting lessons for software engineering in general.



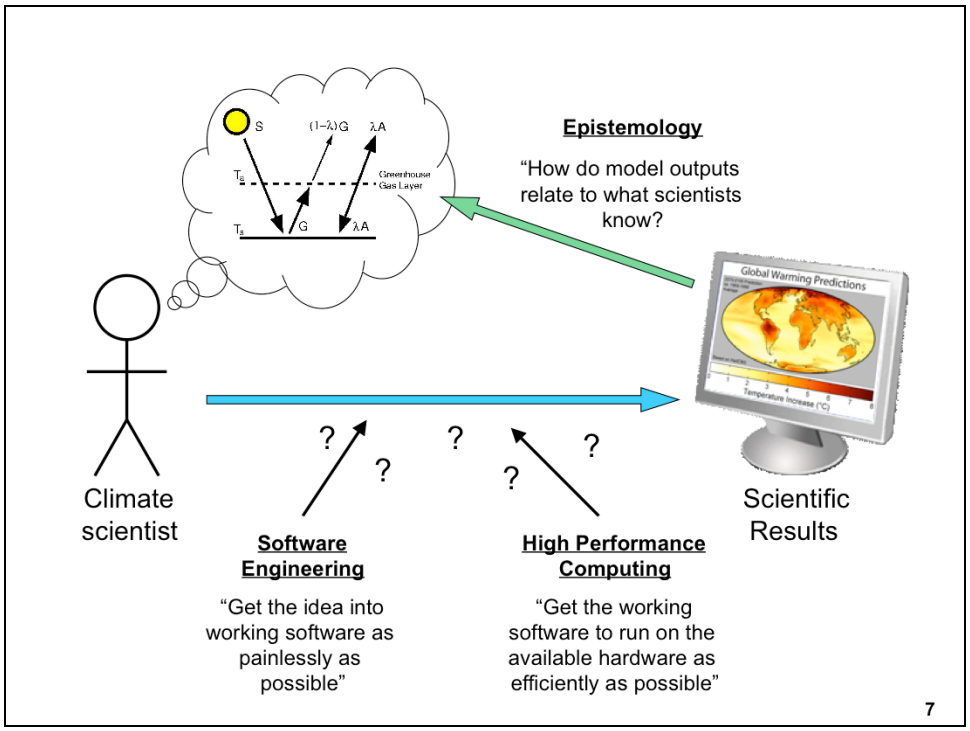
Source: W. F. Ruddiman (2001) Earth's Climate: Past and Future

But before I get to the lessons, I should do a quick overview of what a climate model is. A very simple climate model can be created through the basic energy balance equations that find the equilibrium point between energy absorbed by the earth from the sun, and energy lost from the earth as a warm body radiating infrared radiation into space. But a full climate simulation requires us to understand how the atmosphere and oceans transport heat and mass around the planet. The key idea is to divide the atmosphere and oceans in to cubes, and solve the equations of fluid motion for each cube at each timestep. This gives you the basic dynamics of air and water, and then you can add a representation of radiation (down and up), the properties of water vapour and clouds, dust and other aerosols. Then you can couple this to models of ice sheets, vegetation, rivers, etc, to understand how these systems interact.



Source: NCAR

The result is detailed simulations of the dynamics of weather and climate. For example, this simulation shows precipitation (heavy rain is orange, light rain is white) for a typical August month. Note: it doesn't represent an actual August – it shows typical conditions under current climate. The model can then be used to study how these patterns might change as the climate changes.

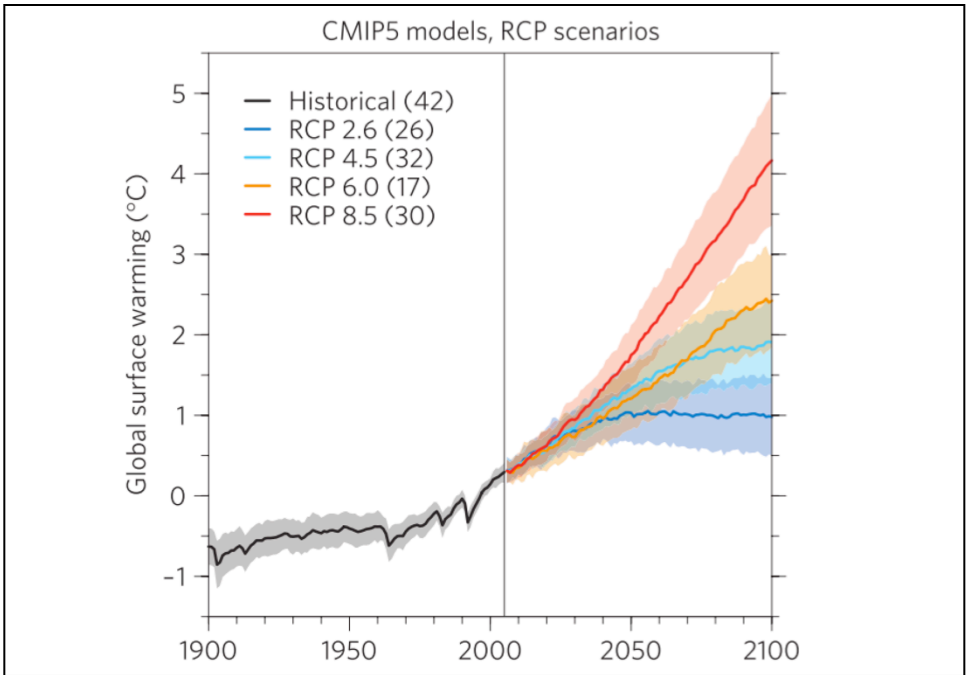


When I set out to do these studies, I began with the question of how the scientists create their models. And especially…

(click) exploring their current software engineering practices, and their use of high performance computing, with the intent of looking for improvements both aspects.

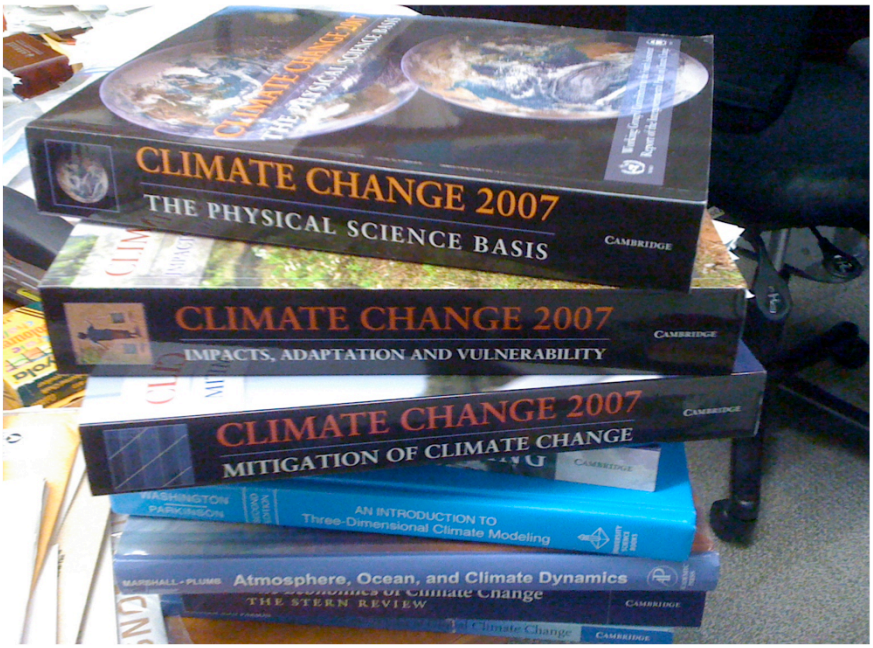
But after several years of investigating these, I came to the conclusion there's a much more interesting question:

(click) How do you explain to the world the epistemological basis for climate model projections? Do the models really capture what we know about the science, and how would we demonstrate that.



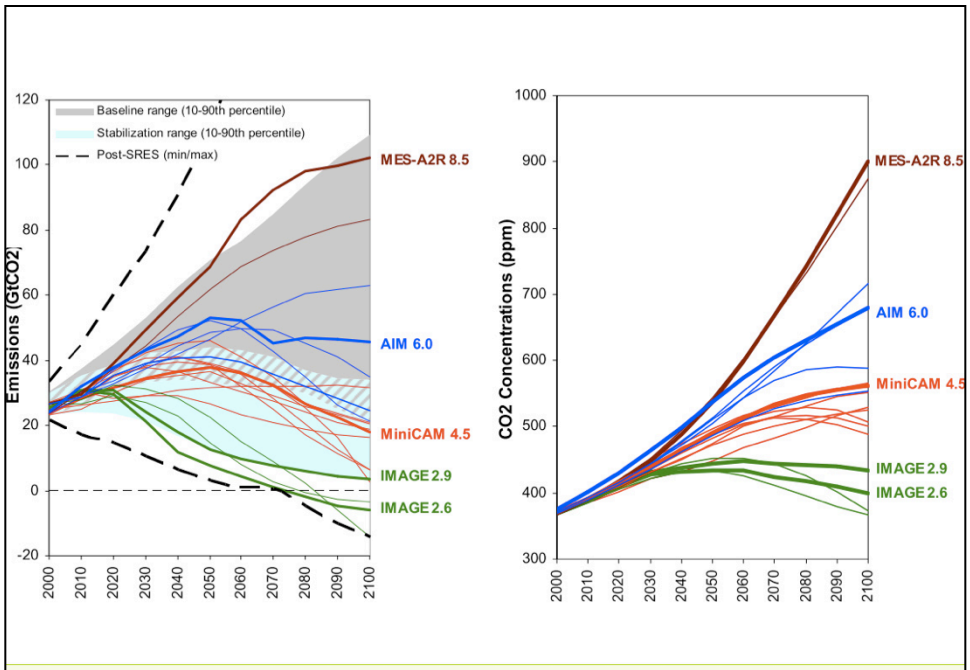
From: Knutti, R., & Sedláček, J. (2012). Robustness and uncertainties in the new CMIP5 climate model projections. Nature Climate Change, (October), 1–5.

The question is particularly important because of the implications of those model projections. For example here's the latest results, currently being analyzed for the upcoming Intergovernmental Panel on Climate Change (IPCC). The four colours represent four possible future worlds. The top two (red and orange) represent a future in which no coordinated climate policy is taken. In the worst case, growth in the use of fossil fuels continues unchecked, while in the more optimistic case, a mix of renewable fuels and slower economic development means lower carbon dioxide emissions. The bottom two (blues) show best and worst case if the world passes serious climate policies to reduce fossil fuel use. Note that numbers in brackets represent the number of different simulation models used for each scenario. About 25 modeling labs around the world build their own models, and they're all run on the same set of scenarios, to provide a probabilistic projection. Also, note that the best case keeps climate change to around +2C above pre-industrial levels (which is much warmer than today), and that this temperature does not drop though the rest of the century.



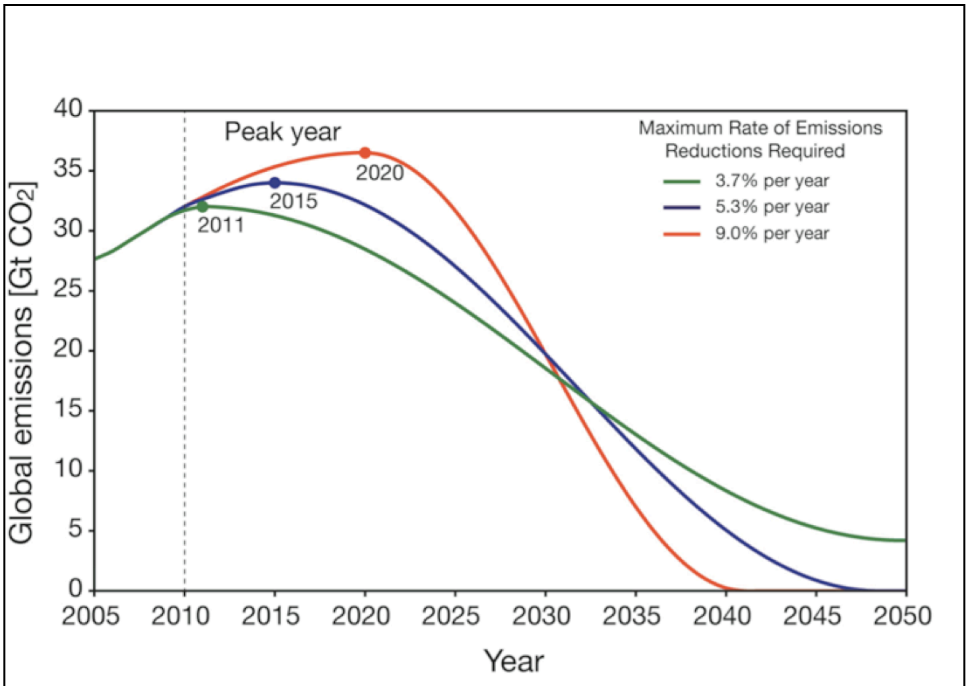
9

By the way, if you're interested in systematic literature reviews, the IPCC assessment is probably the most comprehensive ever undertaken. It runs to 3,000 pages, is printed in three volumes, and is repeated every six years. Volume 1 alone (the physical science basis) had 676 contributing authors, and reviews more than 6,000 published papers.



Source: Moss, R. H., et al. (2010). The next generation of scenarios for climate change research and assessment. Nature, 463(7282), 747–56.

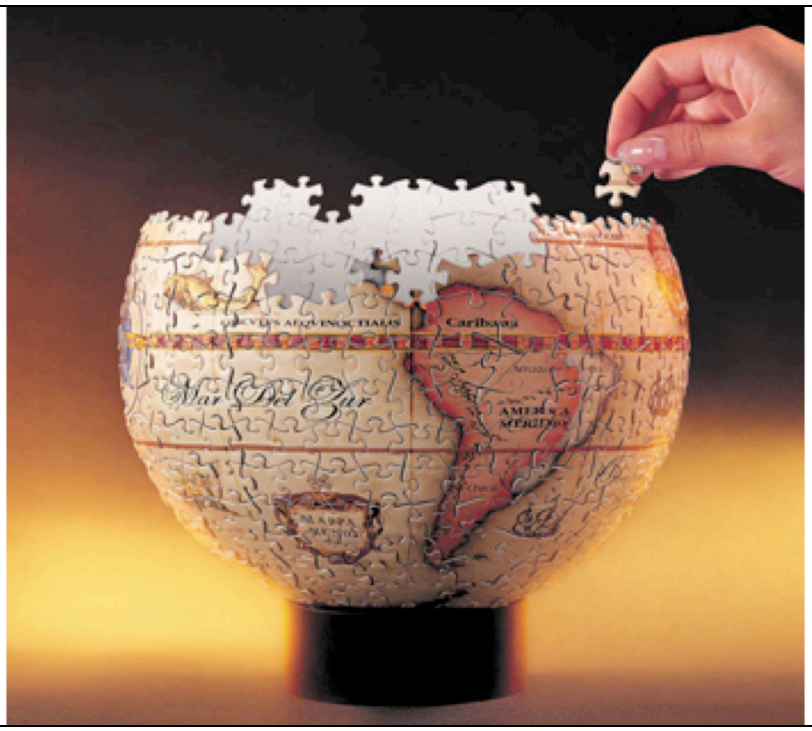
So, my question about how well the models capture the latest science is crucial, because the results are truly earth shattering. To stand any chance of keeping the world below 2C of warming (a target the the UN has committed to) requires use to follow the bottom most emissions path (RCP2.6), which has the world going carbon negative by the 2070s. In other words, it's not enough to stop greenhouse gas emissions altogether – we have to find a way to start removing them from the atmosphere on a massive scale.



Source: The Copenhagen Diagnosis (http://www.copenhagendiagnosis.com/)

Or more simply, we have to keep total cumulative emissions under about ½ a trillion tonnes of carbon. So the later we get emissions under control the faster we have to reduce them.

So how do we know that these projections are any good?



We have to look at how the models are constructed, both from a scientific and a software point of view.

Basic physical equations (>100 years old)

- 1900s: Vilhelm Bjerknes identified the “primitive equations”
  - These capture the flow of mass and energy in the atmosphere;
- Basis of models for weather (and climate) prediction



Zonal Wind:

$$\frac{\partial u}{\partial t} = \eta v - \frac{\partial \Phi}{\partial x} - c_p \theta \frac{\partial \pi}{\partial x} - z\frac{\partial u}{\partial \sigma} - \frac{\partial(\frac{u^2+v^2}{2})}{\partial x}$$

Meridional Wind:

$$\frac{\partial v}{\partial t} = -\eta \frac{u}{v} - \frac{\partial \Phi}{\partial y} - c_p \theta \frac{\partial \pi}{\partial y} - z\frac{\partial v}{\partial \sigma} - \frac{\partial(\frac{u^2+v^2}{2})}{\partial y}$$

Temperature:

$$\frac{\delta T}{\partial t} = \frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} + w\frac{\partial T}{\partial z}$$

Precipitable Water:

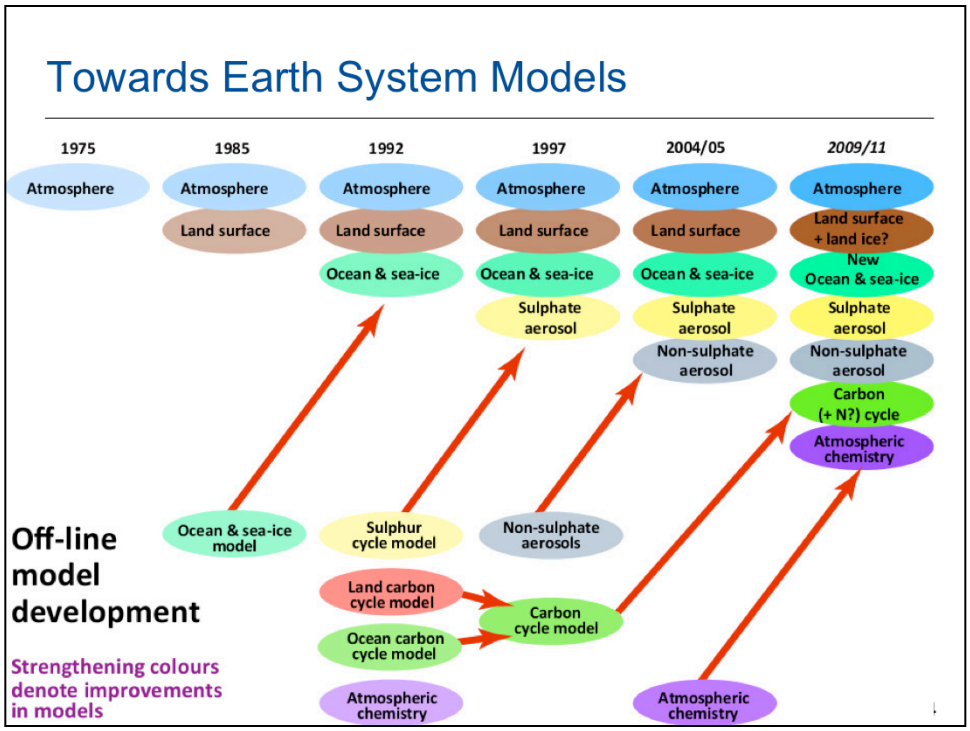
$$\frac{\delta W}{\partial t} = u\frac{\partial W}{\partial x} + v\frac{\partial W}{\partial y} + w\frac{\partial W}{\partial z}$$

Air pressure:

$$\frac{\partial}{\partial t}\frac{\partial p}{\partial \sigma} = u\frac{\partial}{\partial x}x\frac{\partial p}{\partial \sigma} + v\frac{\partial}{\partial y}y\frac{\partial p}{\partial \sigma} + w\frac{\partial}{\partial z}z\frac{\partial p}{\partial \sigma}$$

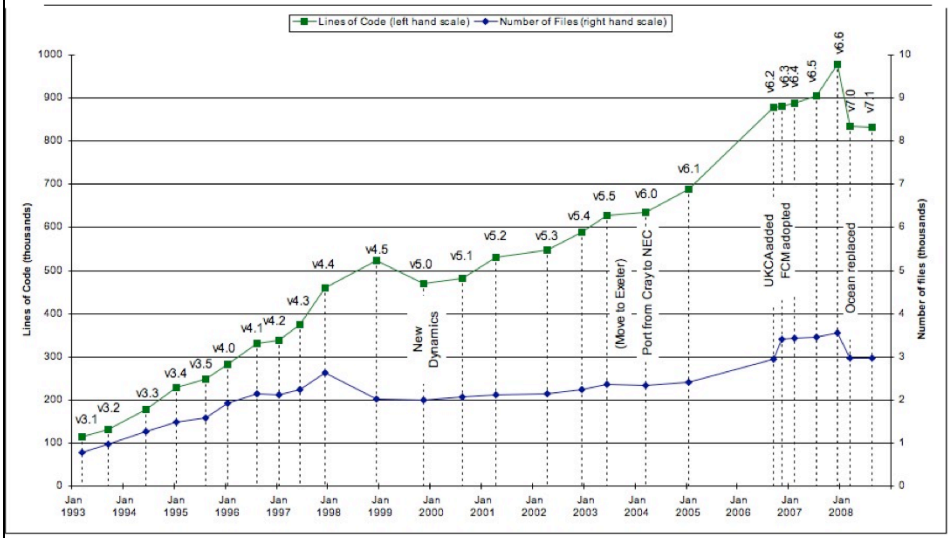
13

First, the basic dynamical equations were worked out over 100 years ago. These capture the thermodynamics of the atmosphere, and allow you to calculate how a weather system develops. You have a set of partial differential equations, and you need some numerical methods to calculate them.



The models have steadily grown as we've learned how to simulate other physical processes and add them to the models. Note that none of this is planned in advance – when someone figures out how to make a simulation work (or how to improve and existing one), and after it's been demonstrated to improve the overall simulation well enough, it gets assimilated into the model. Note also that each of these model components is also a useful scientific simulation on it's own, so it typically has to satisfy two separate uses: stand alone and coupled into an earth system model. The requirements from these two communities are often in conflict.

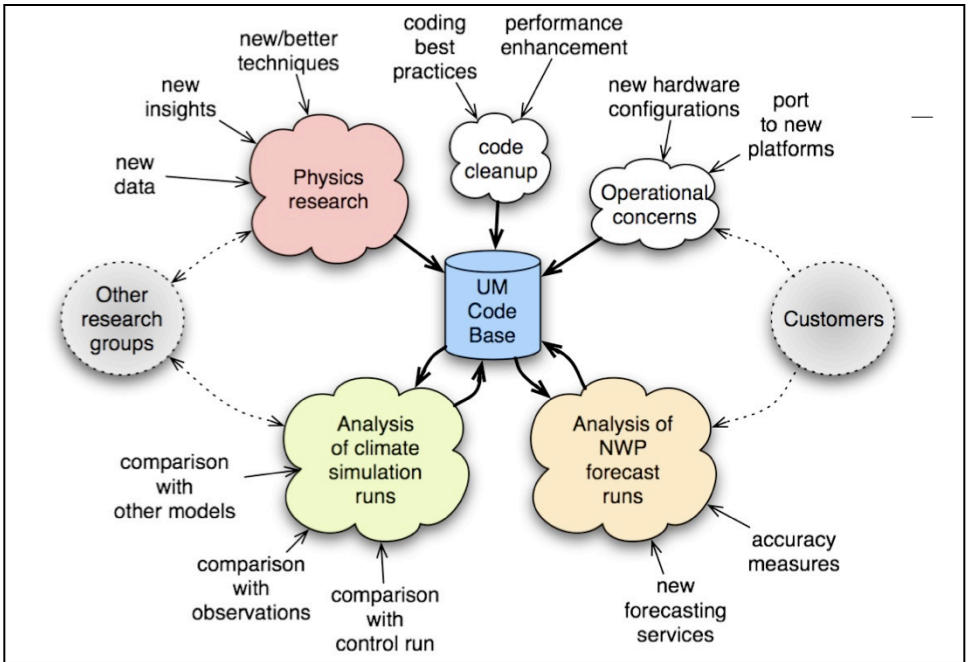
Models are iteratively refined



Source: Easterbrook, S. M., & Johns, T. C. (2009). Engineering the Software for Understanding Climate Change. *Computing in Science and Engineering*, 11(6), 65–74.

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.

This analysis comes from the first study we published, back in 2009.

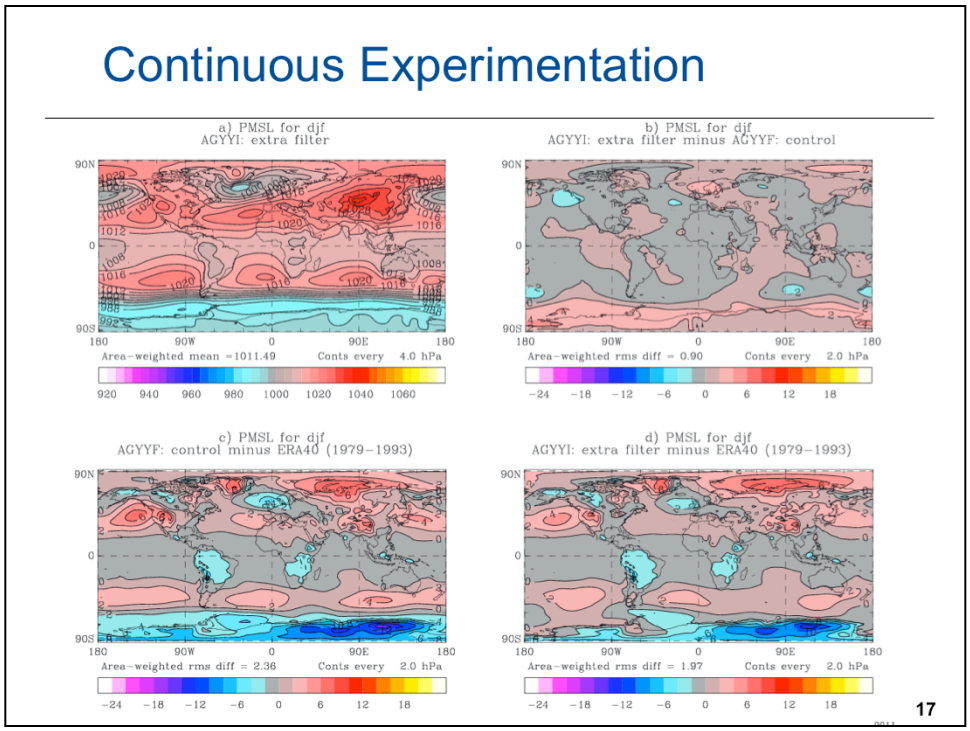


Source: Easterbrook, S. M., & Johns, T. C. (2009). Engineering the Software for Understanding Climate Change. *Computing in Science and Engineering*, 11(6), 65–74.

The development process is essentially a very large agile development (over 50 scientists contribute code to this model for any given release), where the impetus for change comes from a variety of directions.

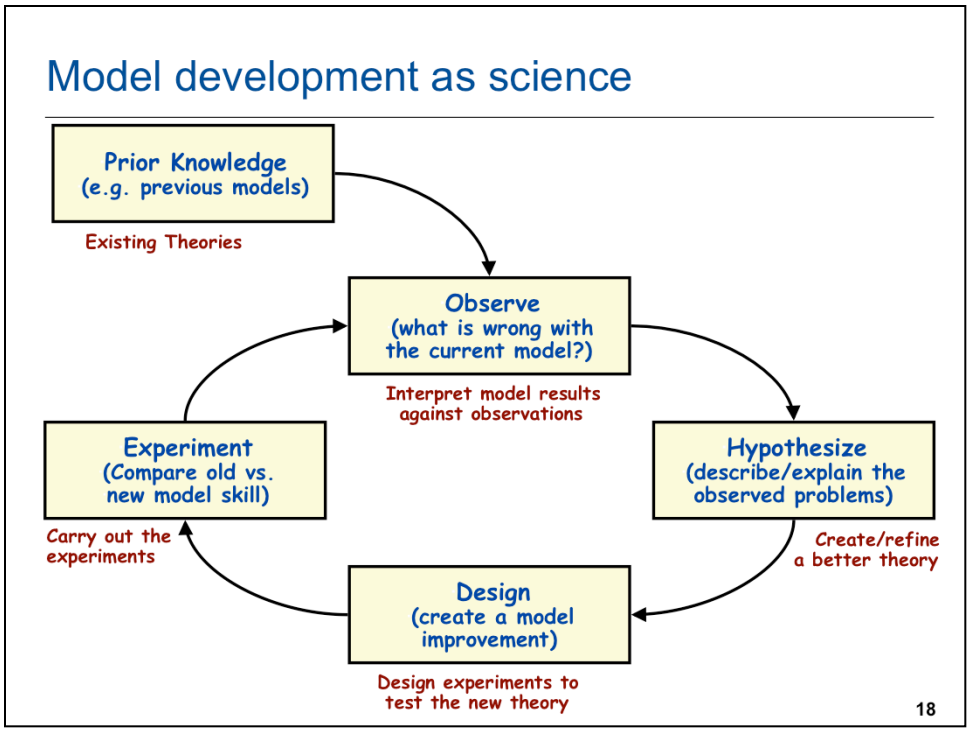
The UK Met office model is unusual in that the same code is used to build operational weather forecasting models, so improvements come from the analysis of forecast skill on weather forecasts, as well as advances in climate science.

Note: I drew the "code cleanup" bubble smaller than the others deliberately. You can think of this as "settling technical debt", but it rarely happens, for very good reasons, which we'll get into shortly.



What's remarkable is that almost every change to the code is run as a scientific experiment. There's hypothesis about how a code change (often very small – a few lines) will affect the simulation results, with improvements expected for particular variables. The hypothesis is formally tested using the existing model as the control condition, and the changed code as the experimental case. This chart shows the result of one such experiment (adding a new polar filter). These outputs show Mean Sea Level Pressure for the months Dec, Jan and Feb. The four displays are:

1) Actual sea level pressure for the new code
2) The difference in pressure between the new code and the control
3) The control compared to observational data
4) The new code compared to observational data.



More abstractly, each change to the code represents a theory-building cycle. In effect, this means the model development process matches the process of scientific discovery.

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). The result is 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).



Now, this is interesting because it means there's a scientific basis for claims that the models are valid. I'm now exploring how to explain this to the general public, so they get some sense of where the scientific results come from, and I'm doing some follow up work, exploring the benefits and weaknesses of this approach. One problem is epistemic entrenchment – design decisions eventually get buried deep within the model, and after a few years are no longer accessible to model users.

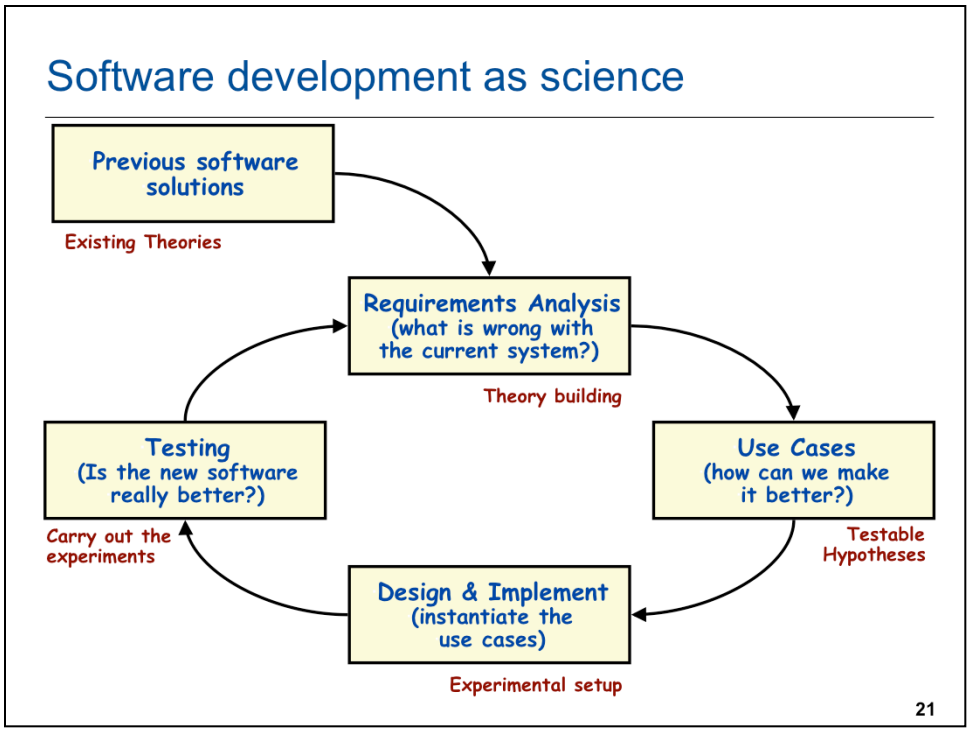
But what I want to talk about now is another possible lesson from these studies.



What if we were to apply this approach to building software to other types of software engineering? Should all software engineering be done like this? It's certainly a provocative question.

We tend to call software engineering an engineering discipline, because there's some notion that we can design solutions using engineering principles. But engineering itself is based on having detailed knowledge of standard types of device (or "machines", if you like), understanding the detailed properties of these devices and the types of problems they can solve. Some SE is like this, where there are standard solutions to common types of problem.

But most of software engineering is deeply embedded in human activity systems, where we don't have a good a priori understanding of what is needed, or even which aspects of a "problem" might be important to different stakeholders. We probably can't completely characterize all the stakeholders anyway.

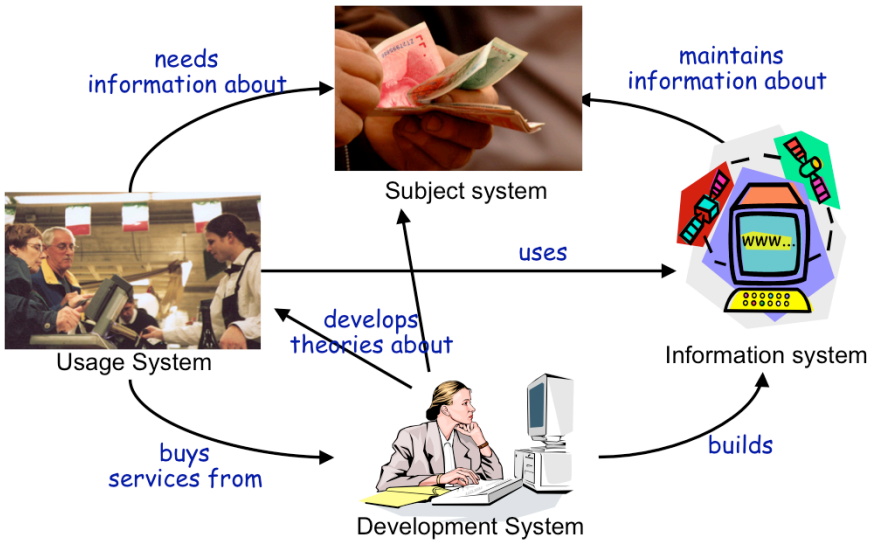


If I treat software development as a scientific process (rather than an engineering process), then it means the requirements are really a theory about the nature of the problem to be solved. Requirements analysis is, in effect, theory building. There's some need, or problem to be solved, in some human activity system. We then test this theory by building the software to implement those requirements, and then see how well we did at solving the problem. Observations of how well we did lead to adjustments to the theory (i.e. an updated set of requirements), and we proceed around the loop.

Note that this corresponds almost exactly to the basic idea of agile software development, except that the vocabulary is different – agile developers don't say they're building a theory of the problem, they just talk about collecting user stories.

But note an important insight: Each time we traverse the loop we're developing better theories. In other words there's a knowledge gain (or at least their ought to be). The software team learns about the nature of the problem by conducting experiments on the user base. So can we measure that knowledge gain?

A software development "system"

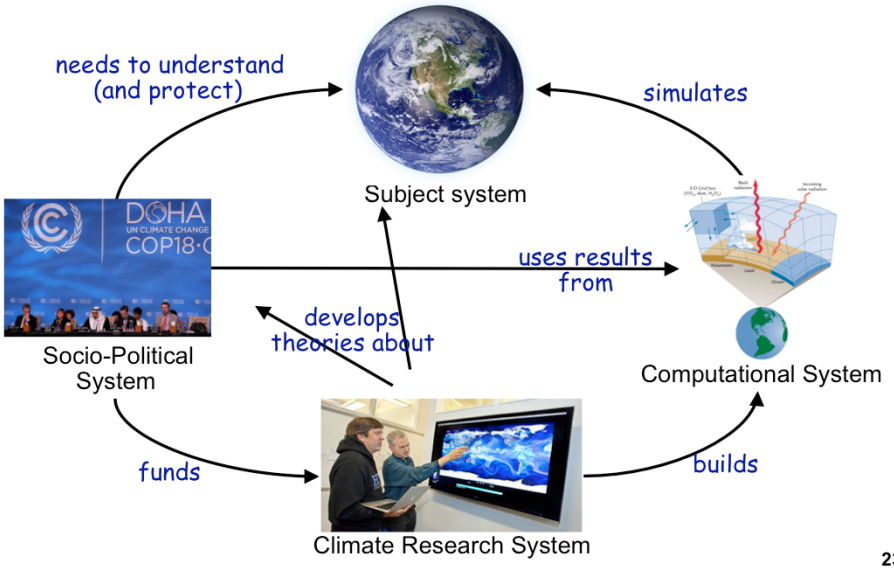


To summarize, we need to think about four different systems.

Over time the developers create theories and refine theories about the subject system and the usage system.

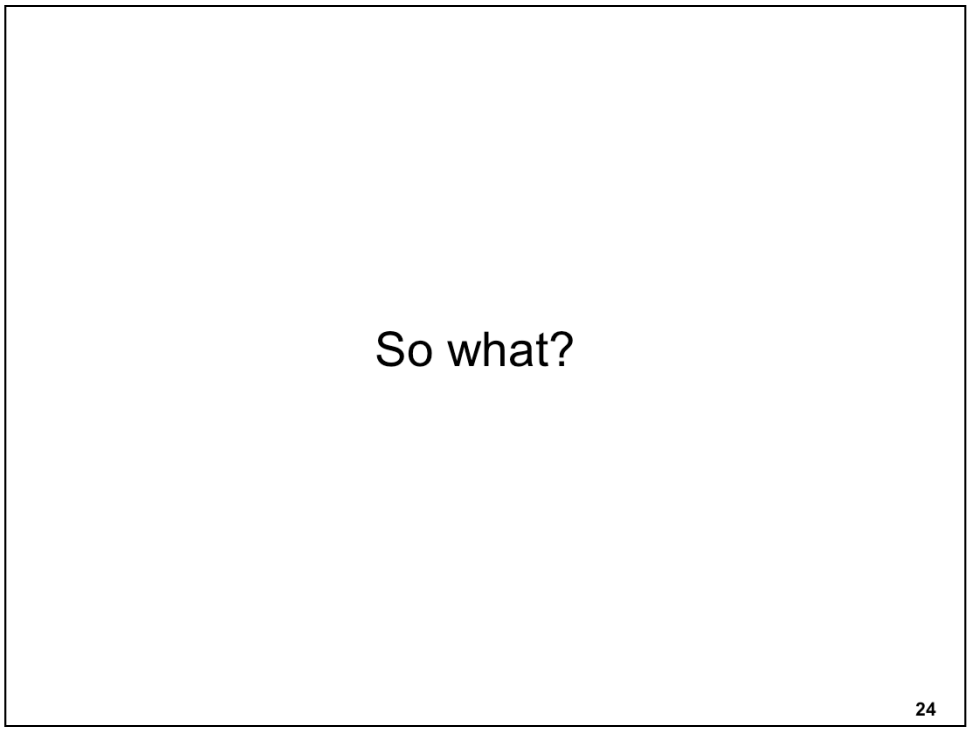
These theories might be about human behaviour, the physical world, the nature of information, the properties of particular technologies.

A software development "system"



23

Which maps onto my model of scientific software development.

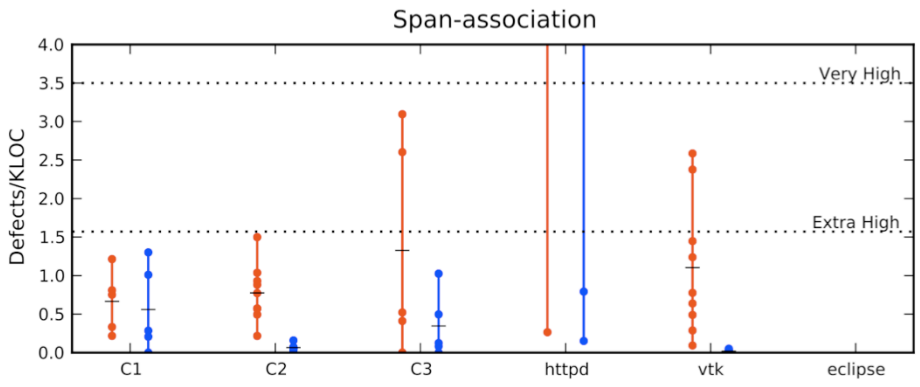


What are the consequences of this view? Well, there are a number of problems in trying to apply our typical research practices in empirical software engineering to the kind of software development that the climate scientists do. I'm going to suggest that these problems are more general, and likely apply to other types of software development.

First, some of our favourite concepts / metrics don't really work.

Second, we need research methods that can help us probe how software teams do this theory building – particularly how they acquire and use knowledge.

Defect Density

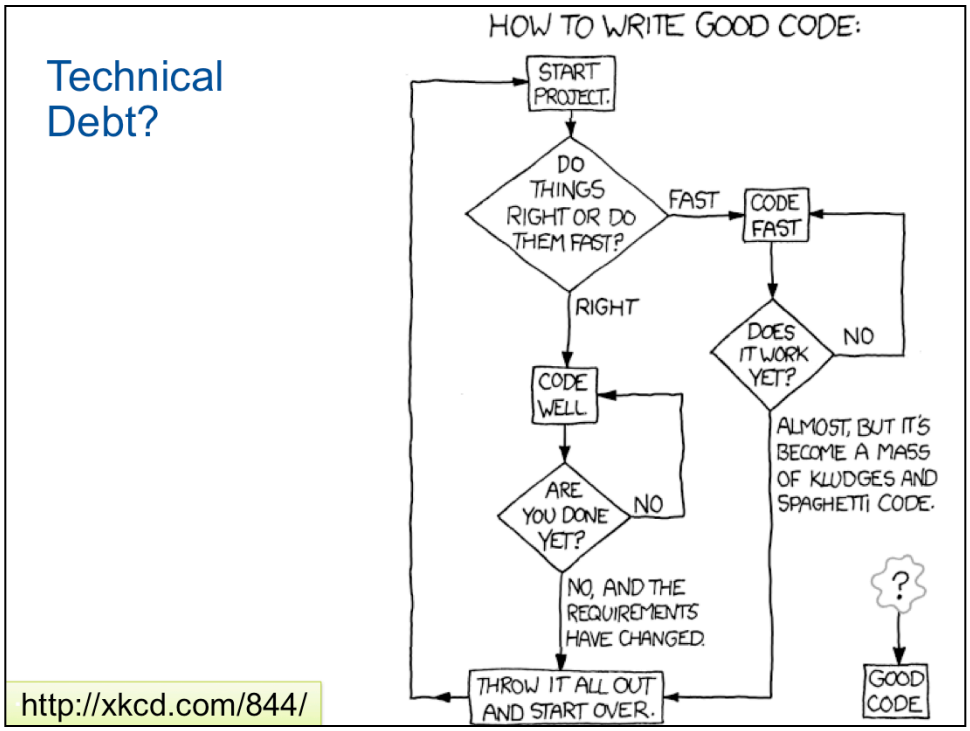


Source: Pipitone, J., & Easterbrook, S. (2012). Assessing climate model software quality: a defect density analysis of three models. *Geoscientific Model Development*, 5(4), 1009–1022.

For example, Defect Density. We tried measuring the defect density of climate models, using standard definitions of 'defect' – essentially a problem that was worth fixing, between one release and the next. Using several different techniques for identifying defects from the code repository and bug database, and several different techniques for assigning defects to releases, we established that climate models have lower defect density than nearly all published reports of defect density in software engineering. Depending on the exact methodology, the models come out somewhere in the "very high quality" to "extra high quality" (based on the work of Pfleeger, Hatton, Boehm, etc).

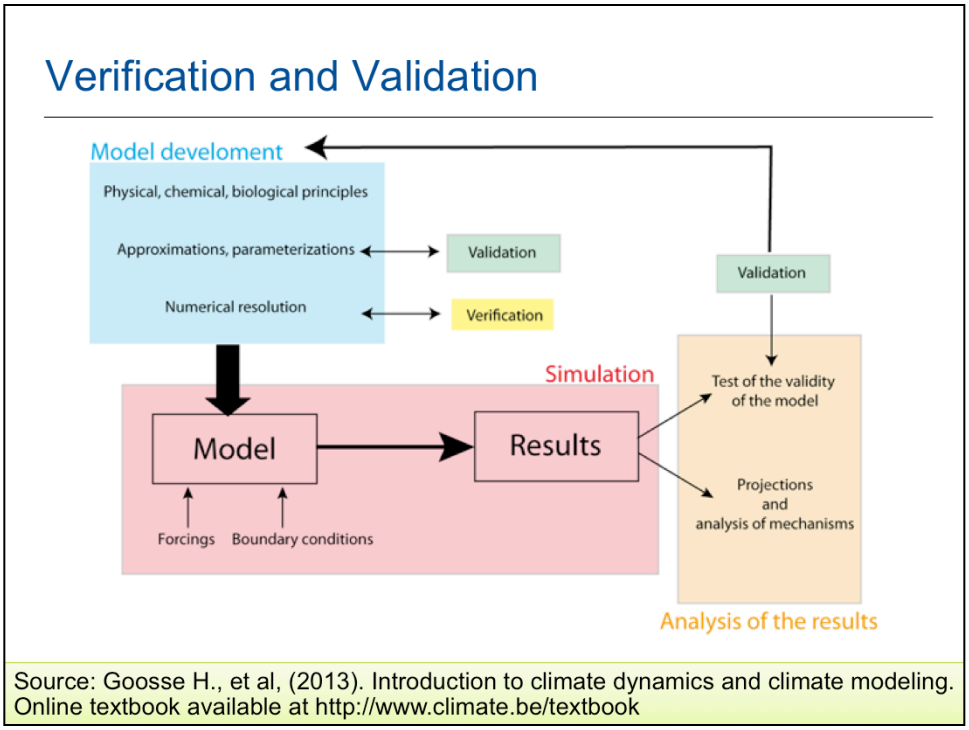
But the problem is, we don't believe it.

If the model is an implementation of an approximate theory of very complex real world processes, what does it mean to say some things are defects, while others are acceptable imperfections? Plus the models are only ever used by experts, who know all the work-arounds. So their tolerance for some kinds of bugs might be very different. If they can get the results they need from the model without fixing what might look to us like bugs, does that make the software defective?



Well, we could try answering that by calling it technical debt instead. But technical debt doesn't really work either, because the software development process is largely opportunistic. Scientific code starts out as individual projects, typically using fast and dirty coding to get a paper ready for publication. The vast majority of this code dies a rapid death after it's first use. It's impossible to predict ahead of time which of these software tools might get picked up and built on by others. So upfront decisions about debt are difficult.

Furthermore, there are many constraints that prevent the kind of software improvement we might expect when paying down technical debt. For example, for the numerical codes used in climate modeling, exact reproducibility (down to the least significant bit) is important, because the underlying processes are chaotic. Because of the way optimizing compilers work, almost any attempt to clean up the code breaks this reproducibility.



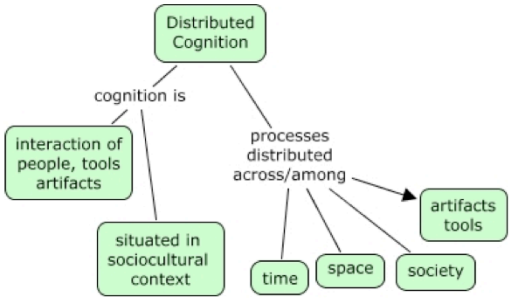
Source: Goosse H., et al, (2013). Introduction to climate dynamics and climate modeling. Online textbook available at http://www.climate.be/textbook

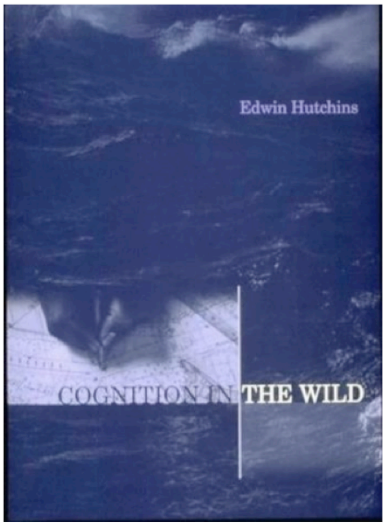
The terms verification and validation don't work the same way either. In software engineering, we normally use these terms within a single development cycle, to refer to how well the code meets it's specification, and how well the specification solves the original problem. In scientific model development, the terms have different meanings.

Verification is almost impossible (except for reduced versions of the numerical algorithms), because there is no analytic solution for the underlying equations, and therefore no test oracle.

Validation is something that now links different iterations of the model development process – it's essentially a theory building step that probes weaknesses in the model skill, identifies reasons for those weaknesses, and suggestions for how to improve them. Rather than telling you how good a particular version of the model it, it just tells you how to improve it.

Distributed Cognition





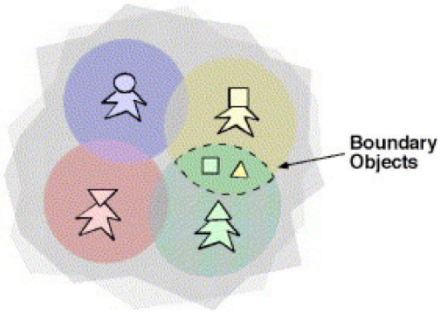
Edwin Hutchins
Cognition in the Wild

28

What about research techniques for studying software teams as knowledge communities?

I would suggest we look to distributed cognition. For example, the classic study was Ed Hutchins ethnographies of how pilots navigate ships into the harbour, where no one team member has complete knowledge of the current situation, but they collaborate without needing it.

Boundary Objects

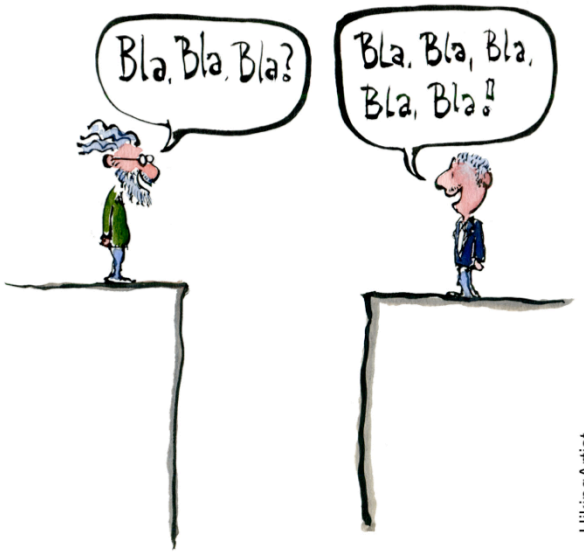




Bowker and Star

Fischer, G., et al. (2005). Beyond binary choices: Integrating individual and social creativity. International Journal of Human-Computer Studies, 63(4-5), 482–512.

The role of expertise

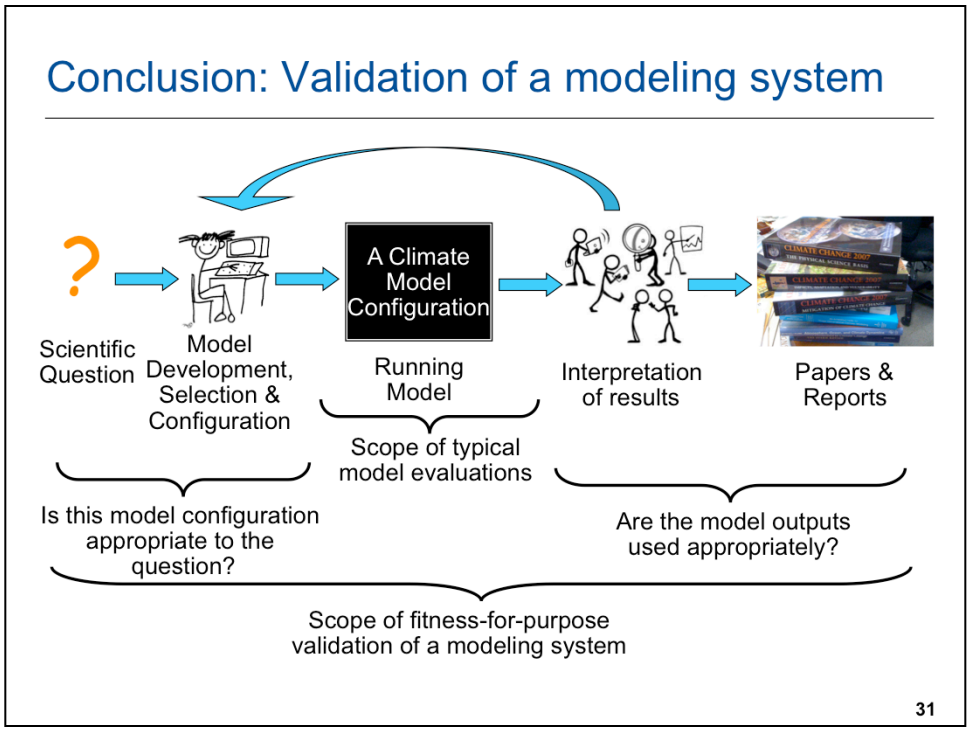


30

Finally, does it matter if the software developers are also domain experts (or become domain experts through the experimentation process I've described?)

For climate models, it appears that it does matter that the scientists write their own code. This eliminates the communication problems.

But this ideal might not be achievable in some software projects. Is it possible to measure the separation between experts and coders?



Okay, time to summarize. One of my key questions is how do we demonstrate that models adequately capture the science. My best answer to this is that this isn't a question you ask of the models in isolation. The models are used by scientists acting as a knowledge community. They understand how to setup and configure models to probe specific question, and they know how to interpret the model results, putting them into context of known strengths and weaknesses of the models. So any question about fitness for purpose (say for a specific policy question) is really a question about the modeling system as a whole, not the models.

Summary

- Climate model instantiates a theory about earth system processes
- Model development = building knowledge about how climate processes work
- Software instantiates a theory about human activities
- Software development = building knowledge about how people and technology might interact
- More focus needed on how software teams build and use knowledge

32

More generally, I've set out an argument that climate models instantiate theories about the climate system, which means that model development is essentially a theory building process.

I've suggested this view can be applied to other types of software. The software instantiates a theory about human behaviour, and then software development is a process of building new knowledge about human activities and how technology might support them.

And I've argued that if we take this view seriously, we need to get to grips with the question of how software teams build this knowledge over time.