

Latent Risks and Dangers in the State of Climate Model Software Development

Shawn M. Freeman
NASA/Northrop Grumman
NASA GSFC Code 610 B28-R135B
Greenbelt, MD 20771
1-(301)-286-7066
shawn.m.freeman@nasa.gov

Thomas L. Clune
NASA
NASA GSFC Code 610 B28-R135B
Greenbelt, MD 20771
1-(301)-286-4635
thomas.l.clune@nasa.gov

Robert W. Burns III
NASA/Northrop Grumman
NASA GSFC Code 610 B28-R135B
Greenbelt, MD 20771
1-(301)-286-9513
robert.w.burns@nasa.gov

ABSTRACT

As the importance of climate modeling dramatically increases due to concerns about global climate change, the quality of model software will come under ever more intense scrutiny. Software defects that alter model predictions could negatively impact important climate policy decisions, and even if detected in time could reduce policy makers' confidence in the science. Effective protection against software defense against errors requires the anticipation of vulnerabilities for every facet of this important work. Unfortunately, existing climate model implementations and associated software engineering practices are inadequate to defend properly against some concerns over defects and untested parameter regimes. Organizations that wish to avoid scrutiny, whether deserved or not, should ensure that appropriate software engineering practices are established with all due haste. In this paper we examine some of the inadequacies of existing software development methodologies and suggest strategies for fundamentally changing the established culture.

Categories and Subject Descriptors

D.2.0 Software Engineering, General

General Terms

Management, Measurement, Design, Economics, Reliability, Human Factors, Standardization, Verification

Keywords

Climate model, climate change, software verification

1. INTRODUCTION

As with many large-scale, complex scientific applications, most climate modeling efforts struggle with the consequences of extremely limited investments in software engineering processes. However, with the continuing rise of global climate change as a major challenge for both public and fiscal policy, the technical and political risks of inadequate software verification of climate models are becoming a major liability. Multi-trillion dollar decisions [5,6] about how to prevent and/or adapt to climate change, are being made by local and national governments as well

as large financial institutions, such as the insurance industry. These decisions are based in large part upon predictions derived from climate models. Estimates of global economic commitments at the recent Copenhagen Consensus on Climate are in the range of \$250 billion annually over 10 years [6]. Even national security policy is beginning to be reformulated to reflect the risk of major political and social unrest driven by consequences of global climate change such as reduced availability of fresh water and/or food in highly populated regions of the globe [3].

Although sophisticated, albeit non-automated, processes have been developed and applied to successfully validate climate model predictions against actual physical observations, relatively little effort has been invested in directly validating various major model subsystems on an individual basis. Software verification at any finer scale is also rare. Model validation generally involves detailed examination of output from an ensemble of long executions and must be repeated each time a major change or correction is made. Because of the extreme difficulty, systematic model validation is generally attempted only after a lengthy (1-2 year) less-constrained development phase. Software defects are typically identified via slow, careful, indirect deductions about discrepancies in the model output. However, such processes are labor intensive and completely inadequate to identify and locate defects that are (a) below the observable threshold, (b) masked by other defects, or (c) only significant for physical conditions that exist in the future and for which there is therefore no observational data to compare against.

Regardless of the actual defect rate in climate models, the lack of formal software development and verification processes leaves climate modeling teams and their predictions relatively defenseless against accusations of unreliable results due to poor quality control, particularly those that relate to future climate regimes. In the broader context, these issues are mitigated by the existence of independent models developed at competing organizations providing scientifically comparable results. However, host organizations, including government agencies, which wish to avoid accusations of poor quality control or the embarrassment of a public retraction of a scientific result, now have a significant incentive to institute appropriate processes as a proactive defense mechanism.

2. CLIMATE MODELING CULTURE AND SOFTWARE DEVELOPMENT

Given the obvious value that stronger engineering processes could bring to climate modeling organizations, it is important to understand the major historical and cultural factors that have

Copyright 2010 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the U.S. Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

Copyright 2010 ACM 978-1-4503-0427-6/10/11...\$10.00.

produced the current state of affairs. Generally these forces are not unique to climate modeling, but rather apply to most scientific models, which are sufficiently complex to require multiple domain experts to implement and understand. Climate modeling is only unique in that its implications have begun to take on importance comparable to industries that have long recognized the importance of robust processes such as transportation, medicine, finance, and spaceflight. Indeed, until quite recently, climate models were used solely for scientific research, with little or no hint of their potential role in public policy. Unfortunately, the limited software engineering practices, which emerged to support basic research, within the modeling community are largely insufficient for meeting the challenges of the newfound role.

Unlike most commercial software, development teams for scientific applications are generally in the form of loose collaborations with relatively weak central management of priorities, coding standards, acceptance criteria and other aspects. Additionally, scientific applications are typically developed by professional scientists (domain experts) with little or no exposure to advanced software practices. Despite these differences most actual software issues are typical of those encountered in other environments. Such issues include accumulated "code debt", poor or unenforced processes, and lack of well-defined standards and/or metrics to ensure software quality.

2.1 Code ownership, individualism, and ad-hoc development

One anomalous feature of software in many scientific modeling groups is that of code ownership. Unlike large or complex applications developed within the commercial and open source sectors where everyone on the project assumes the responsibility of producing maintainable code, the modeling community is more focused on individual goals and producing results. Scientific domain expertise is far less interchangeable leading to a "stovepipe" viewpoint of the model. Thus, a model development team is often better described as a loose-knit community. Common activities performed by professional software engineers such as code refactoring and analysis are often not considered by model developers, and if they are it is considered too complex/time consuming, low or non-priority, or out of scope for individual developers. Code optimization takes precedence over design and elegance, which all too often leads to tangled, complex code. In some cases research-level/prototype code is shoehorned into an existing model by whatever means necessary in order to get results faster. Such activities result in ever increasing amounts of accumulated deficiencies, or code debt, that make future changes more difficult. In addition, most scientists are not well versed in or have been exposed to "clean code" concepts. Even if a scientist attempts keeps long-term maintainability goals in mind they may miss or inadvertently add deficiencies that would otherwise have been caught and corrected by professional software engineers. All of these things are in stark contrast to commercial or open source development efforts where the code must not only operate correctly, but also must meet quality criteria and remain easy to work with.

With model developers focusing more on their own needs and requirements with little, if any, consideration for software engineering practices, a development environment is created where there is no real central authority or consensus over how model development should proceed. Without a central authority or collaborative group's collective decisions to make and enforce

software development policy, scientific software development often takes an ad-hoc/lone ranger approach that is detrimental to the overall maintainability and stability of the code base. Even when there is some semblance of a central authority, this authority is often comprised of science experts, not software engineers. While some standards may be put in place, the disciplines of engineering are often overlooked or ignored in favor of just getting things to work.

The individual-versus-group mentality also creates multiple single points failures with respect knowledge about the code. Lack of documentation, standards, or an authority to enforce those standards leads to cryptic software where only the original developer(s) may fully understand the programming logic, and even they may not remember essential details when another programmer needs to change or use the code. If the developer leaves or is no longer available, a new developer may be forced to waste a considerable amount of time and effort attempting to reverse engineer the logic, or worse, rewriting the entire piece of code.

Not only is ad-hoc development detrimental to the overall maintainability and stability of the model, but it can also introduce issues that are external to the main program. With many scientific models, there is so much complexity introduced by the ad-hoc process that extensive configuration files and scripts are required just to build the program, let alone actually run it. Porting such models to new systems (or even just updating libraries and compilers) can sometimes take weeks to months due to this complexity. Often, extensive access to domain experts is required in order to configure and run the applications correctly. This has the effect of throttling productivity, while creating a drain on resources that could otherwise be better put to use.

2.2 Modelers and engineering apathy

Another aspect that is fairly unique to the modeling community is that it lacks a "user base" in the traditional software sense. In commercial or open source software development, there is usually an external driving force that encourages, if not demands, software quality. Such applications would have few users if the applications were too complex to use or had other faults. Product releases would also be fewer and far between if the existing code base could not be easily extended, re-used, or if bringing on new developers took a large investment of time. Such applications and projects would likely fail and/or be abandoned in favor of better-engineered alternatives. This is in sharp contrast to the modeling community, where fragile, complex, and difficult to use software has become the status quo. The user base is the model developers/scientists themselves. This user community has become complacent and just accepts that models are delicate and difficult to work with. Similarly the developer community accepts that the code base is equally delicate and difficult to modify. This apathy creates and fosters an environment that encourages the modeling community to be it's own worst enemy when it comes to improving their software and associated processes.

2.3 Code confidence

One key area where model development does not differ from other software development projects is the need for methods and measures sufficient to thoroughly test, validate, and quantify the quality of the code. Unfortunately, this is an area where models and model developers fall short. The typical procedure for testing a model is adding new code, compiling the model, and then

performing a lengthy simulation with the new executable to either compare against available data or previous model results. The determination of whether or not the code works is based on the judgment of the model developer/domain expert. These criteria are often not documented, and even less likely to have automated test cases. This is a very poor process for ensuring that complex software is operating correctly. It is difficult to determine whether or not any given aspect of the model behaves, as it should in response to the new code. Software faults can also be hidden by the current run parameters or does not come into play. Other than floating point exceptions, these software faults can be extremely subtle and may not manifest themselves until a specific set of conditions are met.

Whereas these concerns are worrisome enough for the purposes of scientific research, for models that are employed for critical decision-making, this level of uncertainty about quality of the code is simply unacceptable. The idea that models and their results are just for researchers is no longer valid, especially with politicized topics such as climate change. Similar to financial models where million or billion dollar decisions leave little room for error, models used in formulating policy and critical decision-making must be held to an equally high standard of software quality and reliability. However, unlike financial models, science models often lack the rigorous testing and quality assurance that financial models go through before they are employed in activities such as automatic stock and futures trading.

2.4 Inertia, resistance, and reluctance to change

Last but not least, the model development community harbors an unwillingness to change and what appears to be a general distrust of the software engineering discipline. A common view among model developers is that the additional time spent doing actual software engineering versus just programming adds unnecessary overhead that takes away from the "real" work, i.e. scientific research. Another common (and incorrect) view is that the more "modern" programming paradigms (like object-oriented programming) that encourage and/or enforce better support for common infrastructure that is orthogonal to the usual decomposition of responsibilities always introduce significant overhead or complexity in the code, despite numerous research studies that indicate otherwise [1,2]. The prevailing mentality is that current methods and code are "good enough" for the research the modelers wish to perform, and therefore any additional activities just detracts from doing the science. To be fair, many modelers embrace these concepts, but rarely to the point of making significant changes to the investment of resources. Given that a modeling group can often hire 2 postdoctoral researchers or several graduate students for the cost of a single qualified software engineer [4], this viewpoint is not altogether unfounded. The old adage about leading a horse to water remains alive and well in the modeling community.

Unfortunately, there is an incredible amount of inertia within the modeling community in regards to adopting and using better software engineering practices. Even with well-documented advantages from both the commercial sector and open source projects of using better software practices, there is insufficient incentive to effect change. Legacy code bases would require funding (for some models, a significant amount funding) to retroactively apply such practices and paradigms, which science groups currently cannot or will not justify. In the meantime, other

than the self-inflicted losses of productivity caused by code quality, there is no immediate "penalty" to continue with their current methods of operation. Public scrutiny may eventually generate enough concern to effect meaningful change, but for a lot of models that may be a ways off or never materialize.

Put simply, the current methods of model software development are not sustainable. The lack of coherence among the model development communities, apathy towards the development process, and lack of software engineering practices has been and continues to be the source of many issues that continue to plague modeling development. The fact that model developers are often domain experts and not software engineers only contributes to the problems by fostering a self-centered or research-centered approach to development with little concern for the quality of the overall software. These issues have a strong negative impact on productivity, maintainability, and quality of code which are not only costly to the model development community, but could lead to public/political costs and consequences as well. The longer these issues remain unaddressed or ignored, the worse the problems will become, and the more time and resources it will take to rectify the problems.

3. STRATEGIES FOR ENACTING CULTURAL CHANGE

Effecting significant improvement in development processes used in climate modeling in the presence of the entrenched culture described above will require considerable effort on multiple fronts. First and foremost, short-term resource incentives are required to offset the disruption and encourage constructive responses. Further, external expertise must be brought in and provided a certain level of authority to implement change while educating and training other developers. And finally, long-term institutional standards and metrics must be established to ensure that processes and quality are maintained once the initial resource investments have been consumed. Hopefully in that era, the long-term benefits will be more apparent to the community and overt incentives will be less necessary.

Without the augmented funding to incentivize the necessary improvements in software engineering activities, it will be difficult to enact change. Not many modeling groups will willingly divert resources to hire non-scientists because it will be a perceived reduction in the amount of science or model development that can be attained. Additionally, forcefully diverting those resources from existing funds may well result in resentment toward the new software activities.

A better approach would be to provide augmented funding and tie it to well-defined criteria. This ensures that they are used strictly for establishing and applying software engineering practices to the project without stirring deep negativity. With funding in place, establishing a close, collaborative relationship between science/model developers and software engineers will be key for developing standards, processes, and improving code quality. Once processes and standards have been developed and agreed upon, these can be applied to existing code as well as new development.

The most effective facilitator for breaking down the cultural and mental barriers model developers have in regards to software engineering is the practice of pair programming. Pair programming encourages both the engineer and the scientist work together on the code simultaneously. This allows the engineer

gains insight into the code and what the scientist is trying to accomplish, and the scientist gains insight into software engineering. The model developer experiences how software engineering works in practice. This close interaction will allow software engineers to help model developers formulate standards and processes that are efficient and effective in their environment, as well as changing model developer attitudes towards treating software as a goal rather than just a tool.

Once the modeling community is more heavily involved with software engineering, practices such as unit testing, test-driven development (TDD), and continuous integration testing should receive more focus. These practices ensure better code by having robust test suites for most aspects of the code. More importantly, these practices provide essential metrics for gauging the health of the code through test coverage and test results. Unfortunately, the cost of retroactively adding testing to the existing (and large) legacy code bases may be prohibitive. However, certain critical areas of the legacy code can be covered with testing and testing practices can easily be used with new development or refactoring activities.

Regardless of the particular practices that are found to be effective for a given development team, parent institutions have a responsibility to implement and enforce appropriate standards, requirements, and audits to ensure that processes do not regress to an unacceptable level over the long term. Old habits will tend to make an appearance until new ones have taken over. Software policies need not be particularly burdensome, especially if the benefits become more evident to the individual researchers.

4. CONCLUSIONS

Climate modeling has become an extremely important, multinational effort with truly sobering implications. Both the raw significance of the predictions as well as the associated political sensitivities imply that climate modelers have a responsibility to ensure that appropriate measures have been used

to establish the reliability of the forecasts. Although some such measures are well established (e.g. validation against data and multi-model comparisons), the general lack of robust software engineering practices represents a growing risk that is likely to be unacceptable to some organizations. Improvements can be made, but will require substantial resources and persistent pressure for cultural change over the course of several years before sufficient protection will be afforded against such concerns. Fortunately, such investments should also improve productivity of these long-lived modeling efforts, offsetting the near term costs.

5. REFERENCES

- [1] Abrahams, D., et al. "Technical Report on C++ Performance", ISO/IEC PDTR 18015. 11 August 2003.
- [2] Bhakthavatsalam, Sumithra. "Measuring the Perceived Overhead Imposed by Object-Oriented Programming in a Real-time Embedded System." Virginia Polytechnic Institute and State University thesis. 16 May 2003.
- [3] "National Security Strategy" The White House, Washington. May 2010.
http://www.whitehouse.gov/sites/default/files/rss_viewer/national_security_strategy.pdf
- [4] PayScale, Inc. "The PayScale Report".
<http://www.payscale.com>.
- [5] Pearce, Fred. "Top economist counts future cost of climate change." NewScientist. 30 October 2006.
<http://www.newscientist.com/article/dn10405-top-economist-counts-future-cost-of-climate-change.html>.
- [6] Roson, R., "Modeling Climate Change Mitigation Options: A Review of Tol's Contribution to Copenhagen Consensus",
<http://ssrn.com/abstract=1514298>.