Critical Strategies for Improving the Code Quality and Cross-Disciplinary Impact of the Computational Earth Sciences

Johnny Wei-Bing Lin
(Physics Department, North Park University)
Tyler A. Erickson
(MTRI and Michigan Technological University)

Acknowledgments: Thanks to Ricky Rood and Jeremy Bassis at the University of Michigan for discussions.

Lives of Significance and Service
NORTH PARK UNIVERSITY
CHICAGO

Slides version date: February 8, 2012. Presented at the NCAR/UCAR/Boulder-area Software Engineering Assembly conference in Boulder, CO on February 21, 2012.

Outline

- The current insular state of computational earth sciences and why we should care
- Critical strategy #1: Unit testing and code review
- Critical strategy #2: Social coding
- Critical strategy #3: Open application programming interfaces (APIs)
- Examples of cross-disciplinary fertilization possible with open APIs
- Developing the computational earth sciences community to encourage adoption of best practices: Code management
- Possible “first-step” roles for funding agencies and the community.

Bottom line: Adopting these critical strategies will improve the code quality and impact of computational atmospheric sciences.

Insularity of the computational earth sciences and why this is bad

- Symptom of insularity: We use languages no one else uses. Thus:
  - Outside users cannot use or test our code.
  - Code innovations created by others are unavailable to us: Fewer synergies are possible.
- Computational power and tools have exploded outside the HPC community: We can't access the results of that explosion.

Language	Rank	Rating
Java	1	17.913%
C	2	17.707%
C++	3	9.072%

Language	Rank	Rating
Fortran	31	0.381%
Matlab	21	0.573%
IDL	51-100	N/A

(top) The 3 most popular languages. (bott) Popularity of languages used in the computational earth sciences. Data from the TIOBE Programming Community Index for October 2011.

Critical strategy #1: Unit testing and code review results in better code

- Detect faults in code:
  - Code reading, functional testing, or structural testing found, on average, 50% of faults in test code in one study (Basili & Selby 1987).
  - If this is this study's fault detection rate *with* some testing, think what the undetected fault rate would be *without* testing.
- Higher code quality:
  - Structured code reading alone, in one study, yielded 38% fewer errors per thousand lines of code (Fagan 1978).
  - Minimum code quality can increase linearly with the number of tests written (Erdogmus et al. 2005).
- Well-tested code enables code to be used as “black boxes” and thus be more reusable.
- Well-written code matters: “... code is read much more often than it is written.” (Van Rossum & Warsaw 2001).

Critical strategy #2: Social coding can dramatically improve code quality

- Open source "social coding" is a community development method that supports code improvement by lowering the barriers to access and changing.
- Project hosting websites (e.g., GitHub) have robust tools to enable **distributed** (not centrally guided):
  - Forking and merging
  - Code review
  - Identification of code improvements

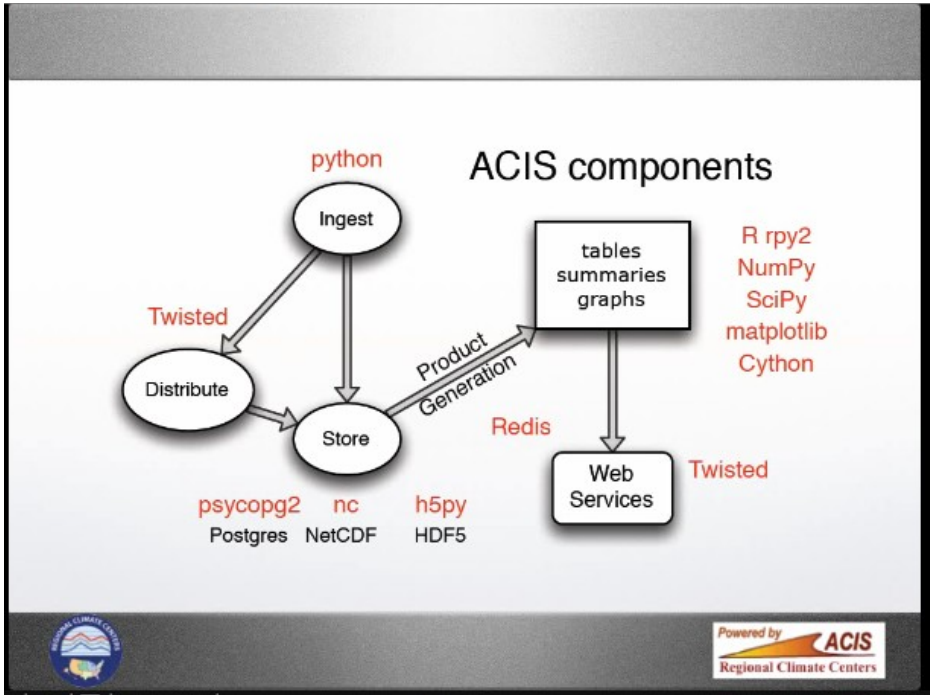
  Program development becomes a very broad-based communal effort!
- Forking a codebase becomes a good, not an evil!:

  "The advantages of multiple codebases are similar to the advantages of mutation: they can dramatically accelerate the evolutionary process by parallelizing the development path." (Stephen O'Grady, 2010)

Critical strategy #3: Open APIs create synergies that increase the impact of code

- Doing good science requires more than just a single tool (i.e., a model) but also includes analysis, visualization, etc.
- The application of atmospheric sciences research to other disciplines (e.g., watershed management) also requires more than just a single tool, including tools not traditionally associated with science (e.g., web services).
- When tools communicate well with each other, you can do a lot more.
- Communication between programs happens through APIs.
- Well-defined APIs make your package usable to many more users and enable unanticipated synergies.

Example of cross-disciplinary fertilization using open APIs: Python and ACIS

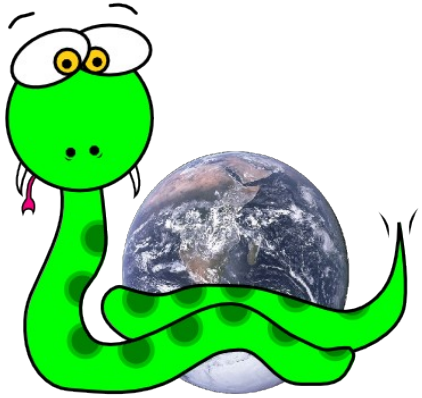


- Problem: Integrating many different components of the Applied Climate Information System.
- Solution: Do it all in Python: A single environment of shared state vs. a crazy mix of shell scripts, compiled code, Matlab/IDL scripts, and a web server makes for a more powerful, flexible, and maintainable system.

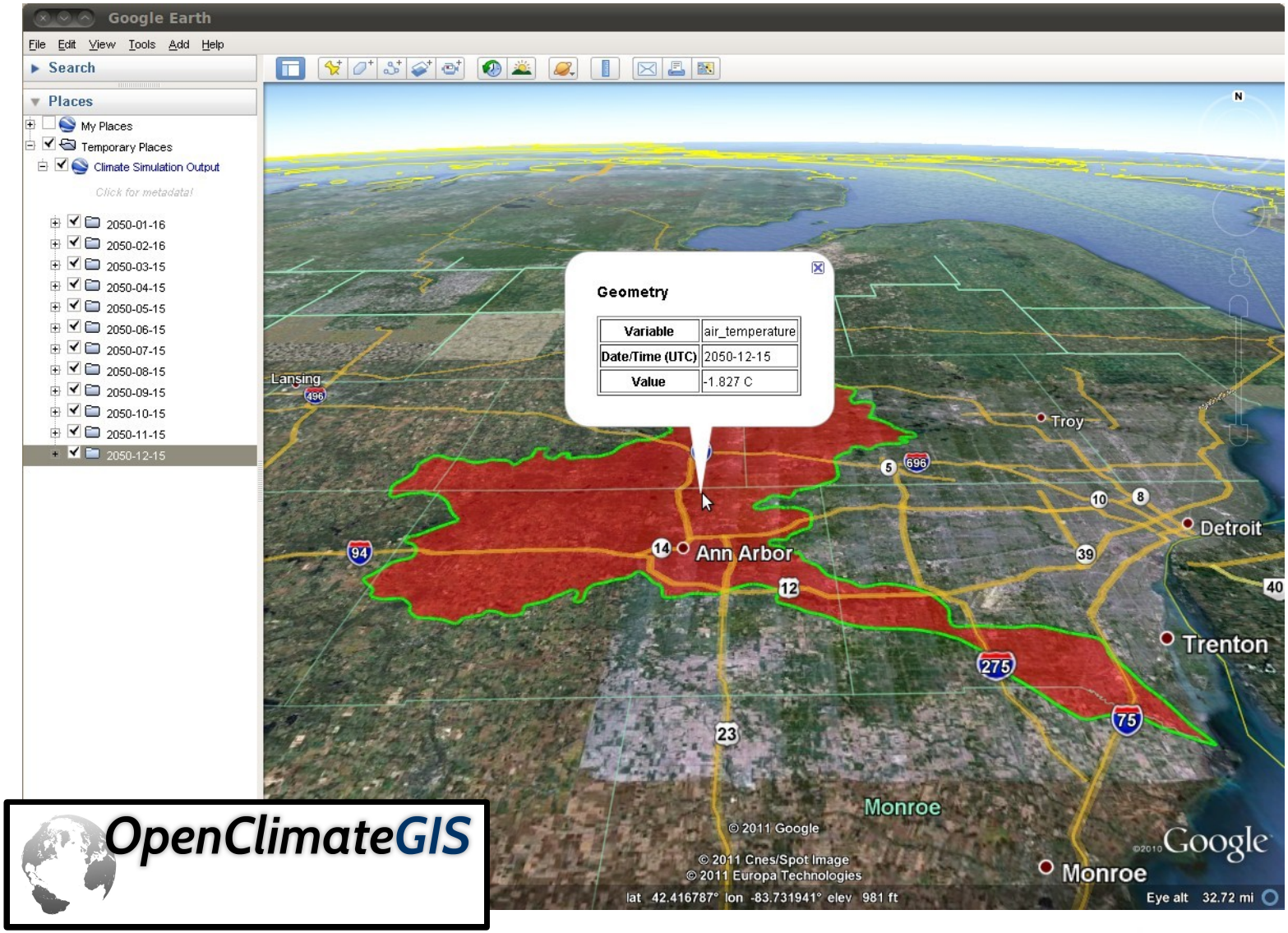
Image from: AMS 2011 talk by Bill Noon, Northwest Regional Climate Center, Ithaca, NY, http://ams.confex.com/ams/91Annual/flvgateway.cgi/id/17853?recordingid=17853

Example of cross-disciplinary fertilization using open APIs: pyKML

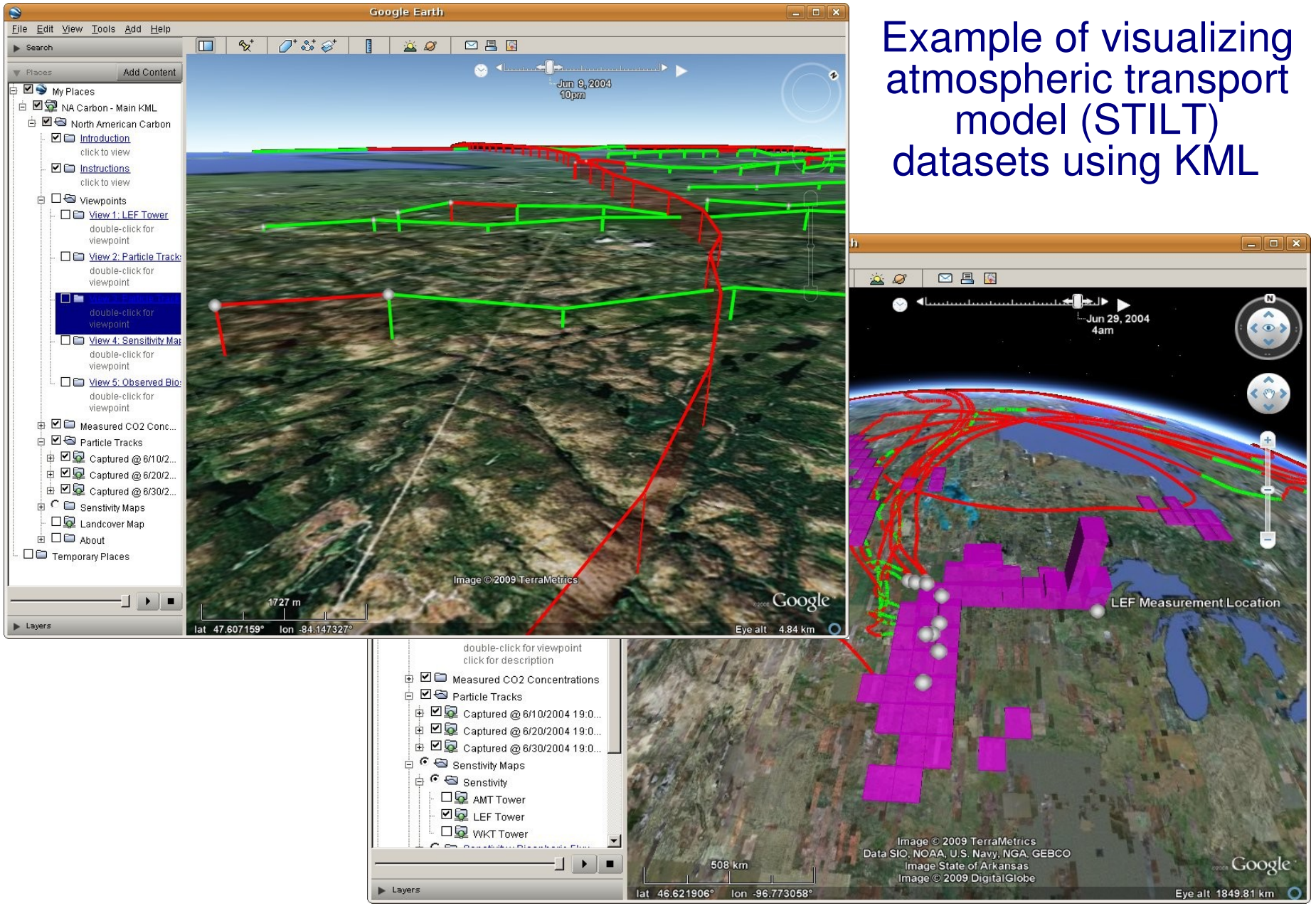
- pyKML is an open source Python library for easily manipulating 3-D spatial + temporal KML documents which provide data to virtual globe applications (i.e., Google Earth).
- Synergies enabled by this open-API:
  - As a Python package, pyKML integrates KML manipulation with data access, geographic/geometric processing, analysis and calculation, web services, etc.
  - pyKML has been used to visualize atmospheric transport modeling and weather and climate modeling datasets.
  - Even Google geo engineers now use pyKML and have recommended it at their own developers conference (Google I/O).



Example of visualizing climate model output data



Example of visualizing atmospheric transport model (STILT) datasets using KML



Developing our community to encourage adoption of best practices

- Goal: Better science through eschewing insularity and encouraging the adoption of software engineering and open-source best practices:
  - Unit testing and code review
  - Social coding
  - Open APIs
- Achieving this goal requires our community rethink how it *manages* code:
  - Code is not just *written,* it can be *used,* by yourself and others.
  - Thus, code is not just a static entity you store but a dynamic entity you manage (or govern).

Seven issues in code management

1) Distribution: How can you make the code available to others?
2) Documentation: How do you describe the code so that others can understand it?
3) Advertising: How do you make sure others can “find” the code?
   - Discover the code exists
   - Realize the code can be applied to their particular problem
4) Instruction: How do you make sure others have the skills that are needed to use the code?
5) Evaluation: How do you learn how your code compares to others people's code?
6) Improvement and feedback: Are their mechanisms to enable users to take your code, use it, improve it, and return those results to the community?
7) Sustainability: Are there (dis)incentives to make code management more (difficult)easy to implement?

The current state of code management

- Most people think code management means distribution and documentation. Thus:
  - The “state-of-the-practice” in earth sciences code management is releasing your code online.
  - The “state-of-the-art” in earth sciences code management is releasing your code online with a manual. ☺
- Ignoring the other aspects of code management results in:
  - Code that seldom gets used by anyone besides the original author.
  - Code that receives limited testing.
  - A lot of reinventing the wheel.
  - Science that is functionally irreproducible.
- But when we consider not just omissions, it's even worse ...

Current practices work against robust code management

- Incentive structure: Scientists are usually recognized for discoveries, not writing great APIs, unit tests, etc., **even if their code enables many others to make discoveries.**
- Opportunity cost: Time writing good, useful (to others) code is time taken away from making discoveries.
- Low community standards: Little public downside to writing untested code.
- Funding: Agencies seldom fund few code management practices beyond distribution and documentation. Even open API development components can be poorly received by proposal reviewers.

Towards better code management

- Technological solutions:
  - Easiest to implement
  - GitHub
  - BuzzData: A Facebook for data
  - VisTrails: Workflow provenance management and “executable papers” that have a paper's computations embedded into the paper.
- Cultural solutions:
  - More difficult to implement but ultimately more influential and effective
  - Metrics of the value of code management efforts to science (e.g., analogous to journal impact factors and citation studies)
  - Lessons from high energy physics: Incentivizing and recognizing co-author #63 on a large and expensive experiment

Possible “first-step” roles for funding agencies and the community

- Cultural incentives: Value quality coding and code advances in addition to scientific discovery
- Financial incentives: Provide resources and requirements to discourage insularity and encourage best practices

Funding agencies roles

- Provide incentives for the publication of model and analysis source code under open licenses.
- Provide incentives for proposals to include a plan for ensuring code quality and openness. This could mean:
  - A structured plan for code review.
  - Source code be asked to pass some minimal suite of tests.
  - Code be hosted on a publicly accessible repository even **during** the project → "real-time code peer-review."
- Support the development of open APIs:
  - This can be an add-on requirement for standard science proposals.
  - Allocate some funding for pure open API development proposals.
  - ESMF is only a step towards this, since scientific computing involves much more than coupling model components.

Community roles

- Expectations: Ask your graduate students or researchers to implement a plan for code review, etc. as part of their regular work.
- Dissemination: Hold seminars, discussions, and courses on software engineering best practices and open APIs.
- Support: Build systems (technological and social) to grow community support for improved coding practices:
  - Training (e.g., AMS 2012 Python short course)
  - Community resources (e.g., pyaos.johnny-lin.com)
  - Social coding (e.g., github.com)
  - Certification

Conclusions

- The time is long past where the computational atmospheric sciences community can practice programming the way it always has.
- Unit testing, structured code review, and social coding can produce higher quality programs.
- Well-written and open APIs can lead to amazing synergies with other disciplines.
- Change requires funding agencies and the computational atmospheric sciences community to support a “new” approach to scientific programming and a holistic plan for code management.