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: December 6, 2011. Presented at the American Geophysical Union 2011 Fall Meeting in San Francisco, CA on December 8, 2011.

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).
- Example of cross-disciplinary fertilization possible with open APIs.
- Agency and community roles to encourage adoption of best practices to break insularity.

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

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.

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!:

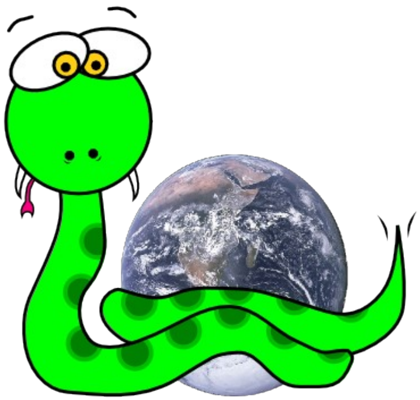
  “The advantages of multiple codebases are similar to the advantages of mutation: they can dramatically accelerate the evolutionary process by parallelizing the development path.” (Stephen O'Grady, 2010)

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

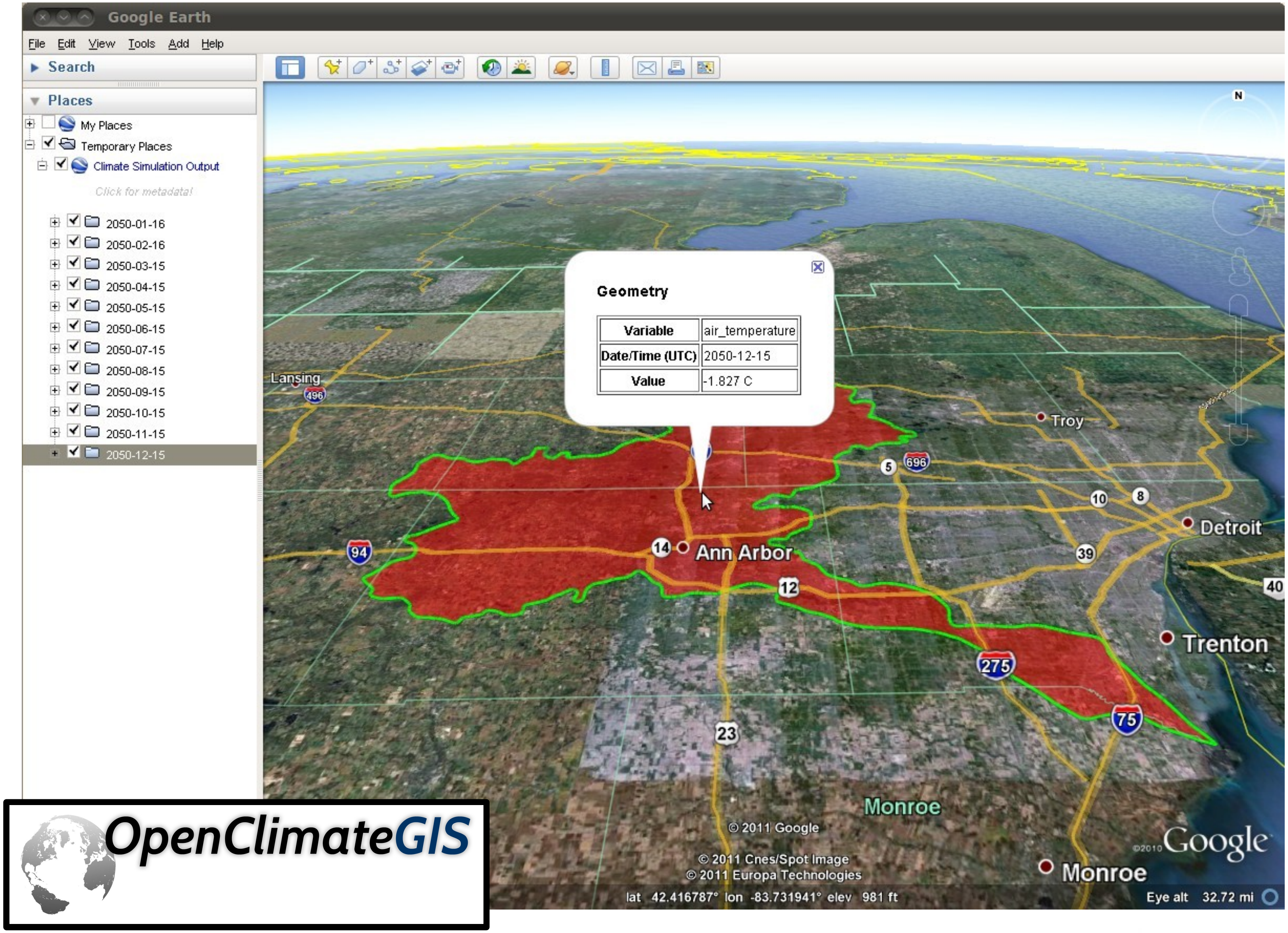
- Doing good science requires more than just a single tool (i.e., a model) but also includes analysis, visualization, etc.
- Applying atmospheric sciences also requires more than just a single tool, including tools not traditionally associated with science (e.g., web services).
- When tools communicate 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: pyKML

- pyKML is an open source Python library for easily manipulating 3-D spatial + temporal KML documents which provide data to virtual globe applications (e.g., 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



Funding agency and community roles

- 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 the goal:
  - 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 agency 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 $\rightarrow$ “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 atmospheric sciences HPC 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 atmospheric sciences HPC community to support a “new” approach to scientific programming.