Open development

Experiences with MOM6

[Shared|Open] [source|development]

- Many science codes in earth system community are “shared source”
    - Occasionally release code
    - Controlled evolution of code
- “Open source” (not what many of us were really doing this)
    - Decentralized
    - Freely available
    - Encourage collaborative development
- “Open development”
    - Same ambitions as “open source”
    - All development is visible
        - deemphasizes the release process

Topics

1. Attitude
    - Why are you doing this?
2. Organization
    - How to work with others?
3. Testing
    - How to make it work?
4. Intellectual property concerns
    - What is the risk?

Attitudes

For original/main developers:

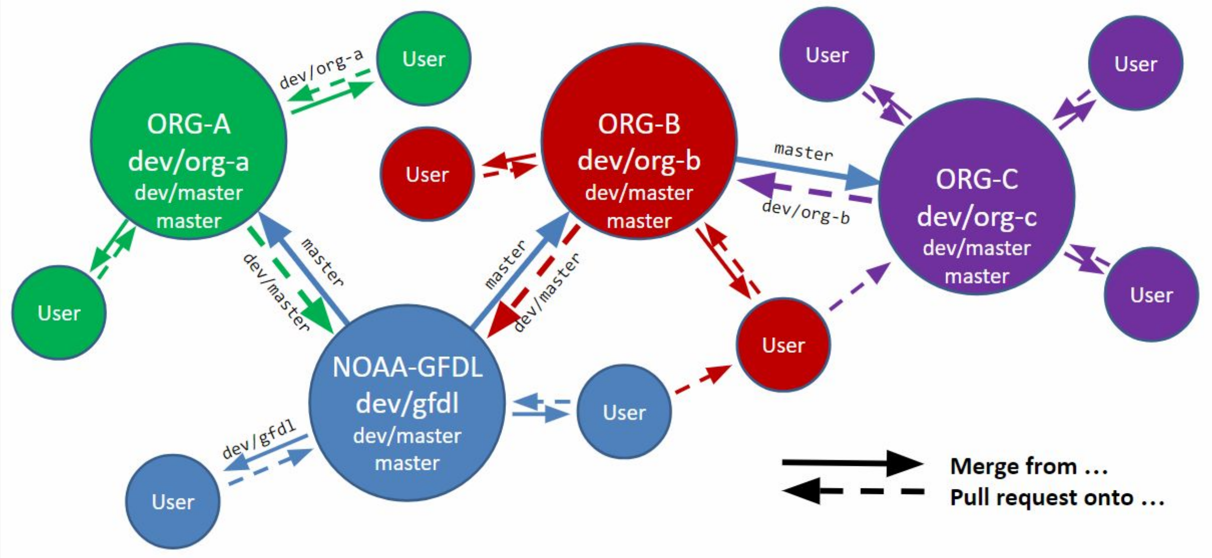
- The main/original/your code **is** imperfect
  - Code is always guaranteed to have bugs
  - Style may be wrong or non-uniform
  - Documentation might be wrong/missing/incomplete
- Contributed code might be even less perfect
  - Better that contributions arrive rather than model be stagnant
- Motivated by desire to collaborate
  - Not to dominate

For contributors:

- Try to understand what the main developers are looking for
  - Read the documentation
  - Look at code examples (especially the recent code changes)
- Main developers will help via feedback
  - You might get it wrong the first time
  - There are no judgements being made
- Consider other users point of view
- Documentation needs contributions too!

MOM6 organization

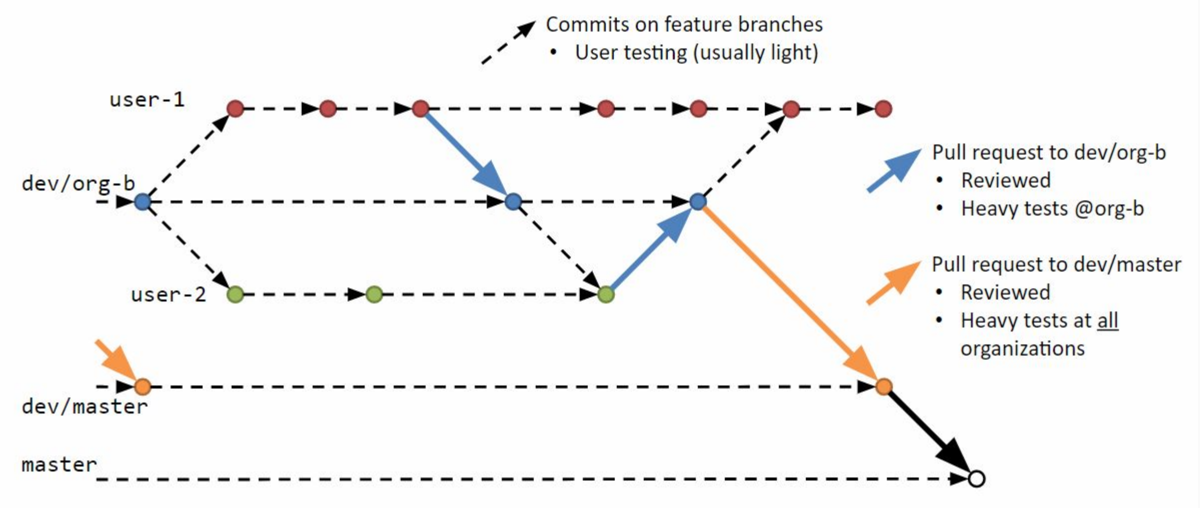
- Each fork develops on its own branches
- Coordinate via a “master” and “dev/master” branch
  - Always in sync
  - Exact same history
- “dev/master” updated via pull requests
  - Evaluated by all partners
  - Orgs define their own tests
- Hierarchical
  - Users fork from/request to organization forks
  - Organizations



- Coordination branches (master and dev/master) make all major forks equal
- Due to the parent/child nature of GitHub forks NOAA-GFDL/MOM6 currently appears as the center hub
  - Not where we want to be
  - Would like to follow NEMO in forming consortium to govern
- Divergence is always a possibility
  - IMHO, it would not be a failure if forks diverge
  - Currently we are all motivated to avoid divergence

All development managed through pull requests

- Core developers are not privileged
- Everyone works on user forks
- dev/gfdl evolves only through pull requests
- Pull requests to dev/gfdl
  - Reviewed
  - Tested heavily at GFDL
  - Daily
- Pull requests made to dev/master only from dev/org
  - Reviewed by all orgs
  - Tested heavily by all orgs
  - Monthly to bi-monthly


Commits on feature branches
• User testing (usually light)
user-1
dev/org-b
user-2
dev/master
master
Pull request to dev/org-b
• Reviewed
• Heavy tests @org-b
Pull request to dev/master
• Reviewed
• Heavy tests at all organizations

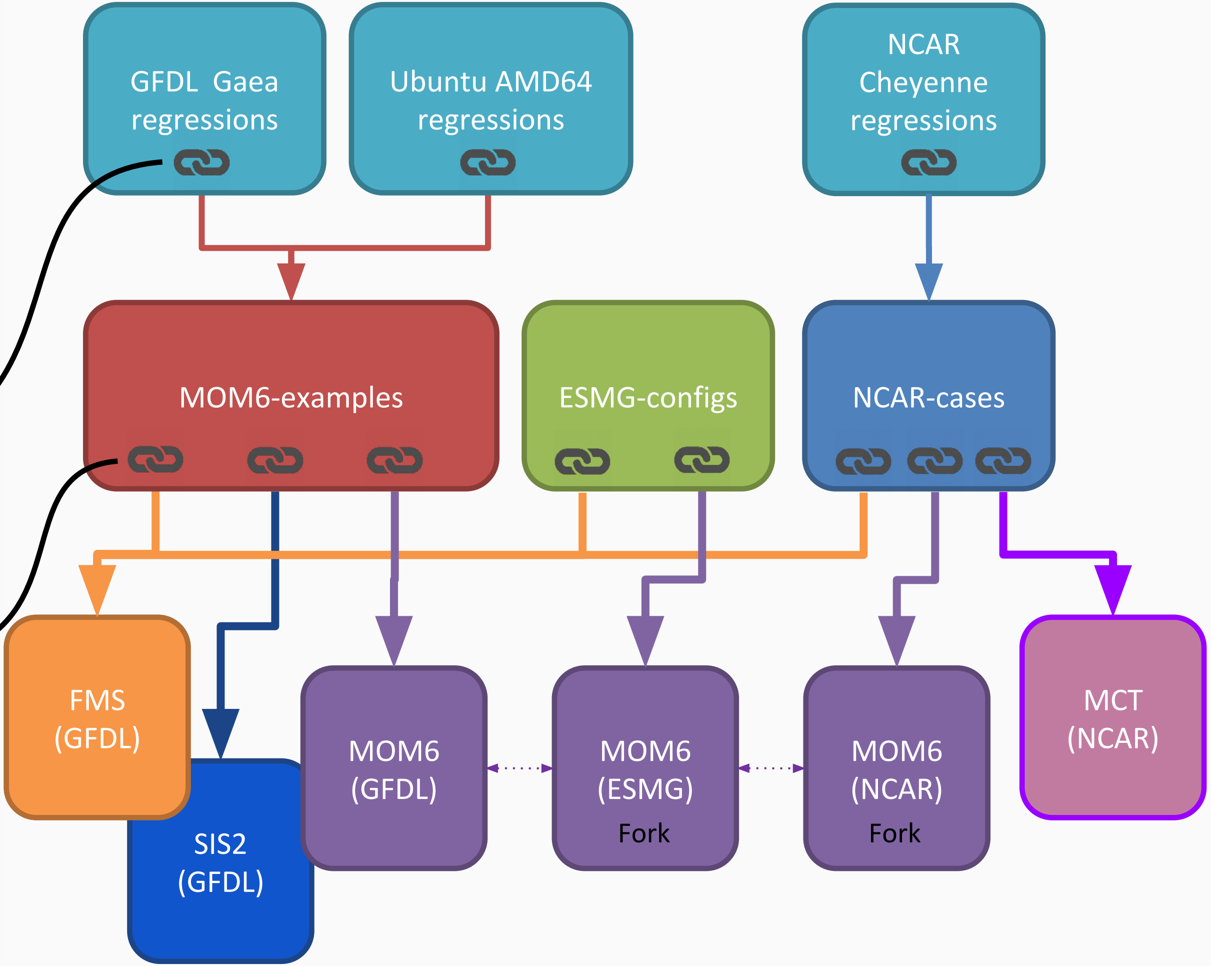
- All users work on their own forks
  - When devs were making branches on the main repo the branches become a mess
  - User forks are under user control
    - No rules - let people work their own way
    - We do provide recommendations for their sake (people learn best the hard way)
- Pull requests to dev/master and master restricted
  - Keeps evolution hierarchical
  - Lowers frequency/burden of managing master

Version control everything

Layered repositories using sub-modules

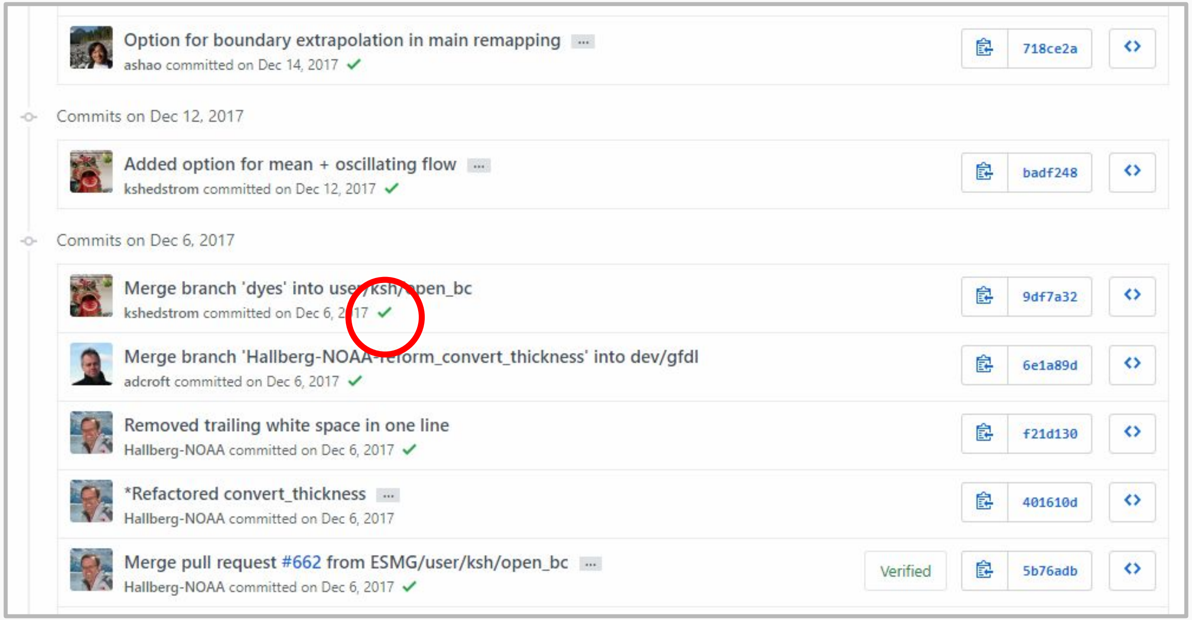
- Regression results
  - Output from regression tests
    - **Platform dependent**
  - Records specific version of configurations
- Configurations
  - Input files (parameters)
  - Records specific versions of source
  - Including URLs (for forks)
- Source for MOM6, FMS, SIS2, …
  - Pure source code (+ packages)

Version controlling everything takes you a long way towards reproducible science



Continuous integration (1/2)

- User can test/develop as they see fit with their resources
- All pushes to GitHub undergo “light” testing using **Travis-CI**
  - Tests fit on single core
  - Must be fast
    - 10-15 mins
  - Cannot test fully deployed models with data etc…
- Early feedback to users
  - Including style!



Option for boundary extrapolation in main remapping …
ashao committed on Dec 14, 2017 ✓ — 718ce2a

Commits on Dec 12, 2017

Added option for mean + oscillating flow …
kshedstrom committed on Dec 12, 2017 ✓ — badf248

Commits on Dec 6, 2017

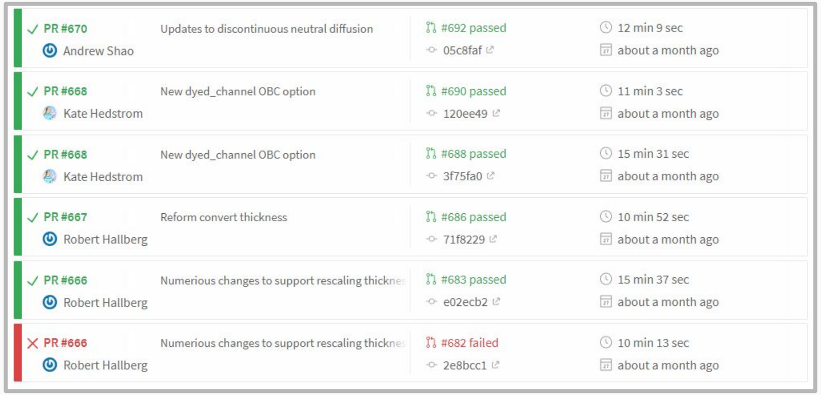
Merge branch 'dyes' into user/ksh/open_bc
kshedstrom committed on Dec 6, 2017 ✓ — 9df7a32

Merge branch 'Hallberg-NOAA-reform_convert_thickness' into dev/gfdl
adcroft committed on Dec 6, 2017 ✓ — 6e1a89d

Removed trailing white space in one line
Hallberg-NOAA committed on Dec 6, 2017 ✓ — f21d130

*Refactored convert_thickness …
Hallberg-NOAA committed on Dec 6, 2017 — 401610d

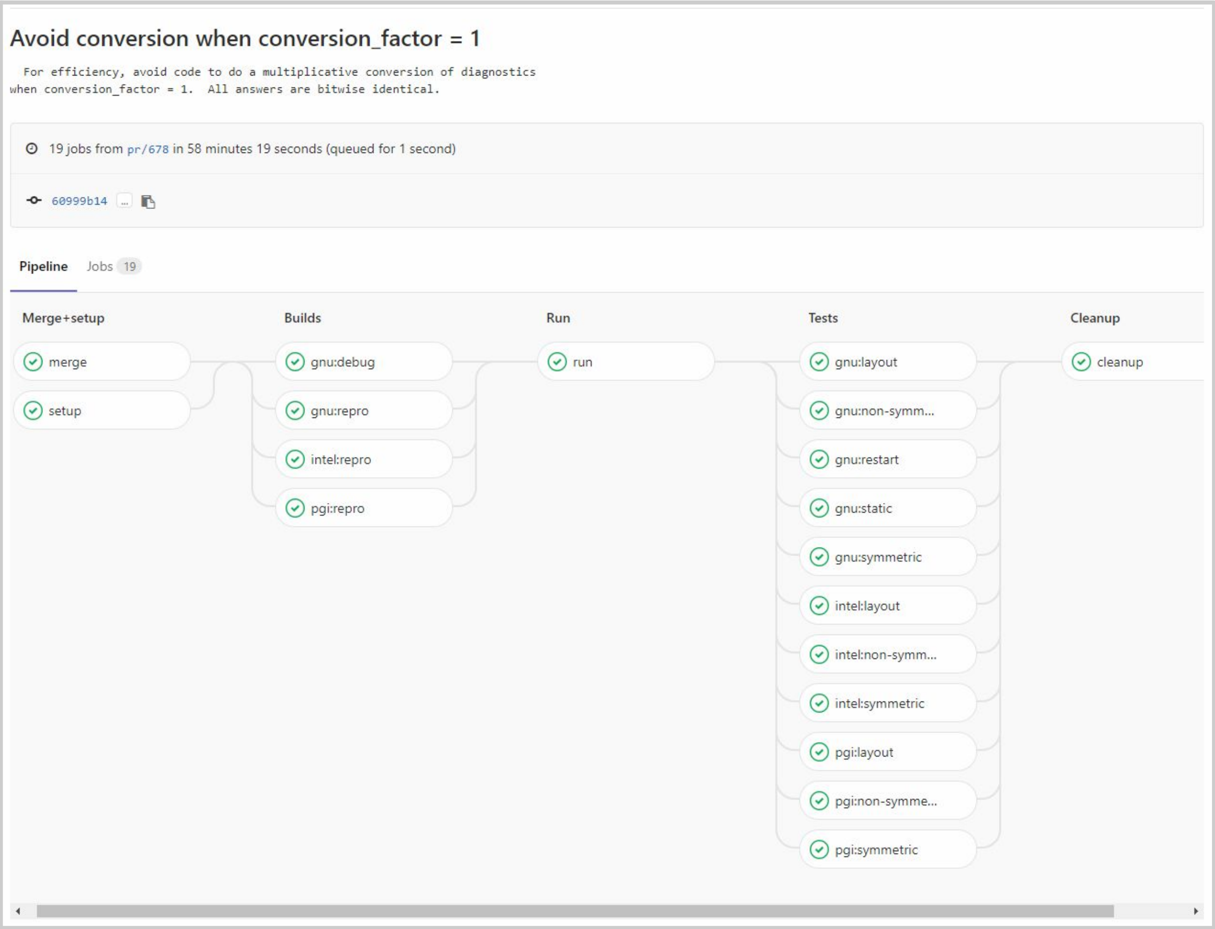
Merge pull request #662 from ESMG/user/ksh/open_bc …
Hallberg-NOAA committed on Dec 6, 2017 ✓ — Verified — 5b76adb



✓ PR #670 Andrew Shao	Updates to discontinuous neutral diffusion	#692 passed 05c8faf	12 min 9 sec about a month ago
✓ PR #668 Kate Hedstrom	New dyed_channel OBC option	#690 passed 120ee49	11 min 3 sec about a month ago
✓ PR #668 Kate Hedstrom	New dyed_channel OBC option	#688 passed 3f75fa0	15 min 31 sec about a month ago
✓ PR #667 Robert Hallberg	Reform convert thickness	#686 passed 71f8229	10 min 52 sec about a month ago
✓ PR #666 Robert Hallberg	Numerious changes to support rescaling thickne	#683 passed e02ecb2	15 min 37 sec about a month ago
✕ PR #666 Robert Hallberg	Numerious changes to support rescaling thickne	#682 failed 2e8bcc1	10 min 13 sec about a month ago

Continuous integration (2/2)

- When a pull request arrives to dev/gfdl
  - Review
  - Submit branch to internal gitlab repo (behind firewall)
    - Invokes job on pipeline
    - Extensive testing
- Post results to GitHub
  - Merge via GitHub



Avoid conversion when conversion_factor = 1

```
 For efficiency, avoid code to do a multiplicative conversion of diagnostics
when conversion_factor = 1.  All answers are bitwise identical.
```

19 jobs from pr/678 in 58 minutes 19 seconds (queued for 1 second)

60999b14

Heavy test harness at GFDL (1/2)

- Regression tests
  - Continuity of solutions using
    - 3x different compilers
      - Gnu, Intel , PGI
    - 3x different memory models
      - Dynamic non-symmetric (traditional)
      - Dynamic symmetric
      - Static (either symmetric or non-symmetric)
    - All with 40-50 test cases

- Reproducibility
  - Across parallel decompositions
  - Across restart boundaries
  - Thread safety
- Compatibility/code quality
  - Compiles & runs in debug mode
  - Code coverage analysis
  - Code style (white space checker!)

To be added:

- **Dimensional analysis**
- **Symmetry under logical rotations**

Heavy test harness at GFDL (2/2)

Testing requires significant resources

- Compilation (full coupled model)
  - ~20 mins/32 cores
- Running optimized executable
  - ~30 mins/1000 cores
- Running debug executable
  - ~2 hr/1000 cores
- Code coverage
  - ~4 hrs/1000 cores

- Documentation generation via doxygen
  - ~40 mins on readthedocs.org

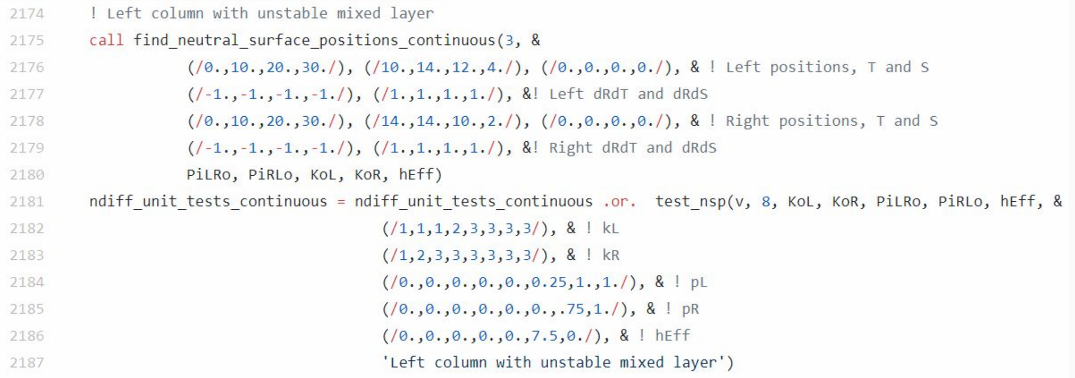
- Heavy harness is **too large** to expect external users to use
  - Likely working on laptop

- Collaborators run different tests for us:
  - Valgrind
    - checks for memory leaks
    - 12 hours/1000 cores
    - (Valgrind doesn't work on our system)
  - Tests of diagnostics

What doesn’t work: testing new contributions

- Our testing insulates us from code changes/contributions that break our configurations
  - If a new piece of code is not triggered then it does not get tested
- Need to figure out how to let a contributor also provide tests
  - Regression tests are platform dependent
    - A user cannot submit correct answers without access to each platform?
  - White paper approach?

- Unit tests are one solution



```fortran
! Left column with unstable mixed layer
call find_neutral_surface_positions_continuous(3, &
          (/0.,10.,20.,30./), (/10.,14.,12.,4./), (/0.,0.,0.,0./), & ! Left positions, T and S
          (/-1.,-1.,-1.,-1./), (/1.,1.,1.,1./), &! Left dRdT and dRdS
          (/0.,10.,20.,30./), (/14.,14.,10.,2./), (/0.,0.,0.,0./), & ! Right positions, T and S
          (/-1.,-1.,-1.,-1./), (/1.,1.,1.,1./), &! Right dRdT and dRdS
          PiLRo, PiRLo, KoL, KoR, hEff)
ndiff_unit_tests_continuous = ndiff_unit_tests_continuous .or.  test_nsp(v, 8, KoL, KoR, PiLRo, PiRLo, hEff, &
                              (/1,1,1,2,3,3,3,3/), & ! kL
                              (/1,2,3,3,3,3,3,3/), & ! kR
                              (/0.,0.,0.,0.,0.,0.25,1.,1./), & ! pL
                              (/0.,0.,0.,0.,0.,0.,.75,1./), & ! pR
                              (/0.,0.,0.,0.,0.,7.5,0./), & ! hEff
                              'Left column with unstable mixed layer')
```

- ...but a lot of code does not fit a unit test approach
- Checking of doxygen-based documentation in new code

Intellectual property

- Users can still work in private
  - I’ve no idea how many do
- Is there a risk?
  - Who has time to monitor your contributions, understand your code, implement their own version, write a paper about?
    - And who would be stupid enough to steal when there is a record of the idea on GitHub?
  - … but yes, there is a risk. I don’t think it has happened to us yet.
- A better solution?
  - Return to old way of doing things, i.e. release code after publication :(
  - Or change the “career system” to reward development as well as papers… :)

Final remarks

- MOM6 has definitely benefited from open development
    - Significant improvements via new code / numerous bug fixes / analysis of configurations
- **Automated testing and continuous integration is essential**
    - As number of external contributors grew, burden on core developers grew
    - Automated testing has removed the time-consuming aspects
        - Only remaining burden is social
            - Some issues need delicate handling
    - AT+CI has also improved developer workflow
        - Core developers follow same procedures as everyone else
        - Better communication
        - Inhibits bad habits, stops shortcuts, fewer mistakes