How do we Help Scientists Embrace their Inner Research Software Engineer (RSE)?

From a Land Component Model (CTSM) Perspective

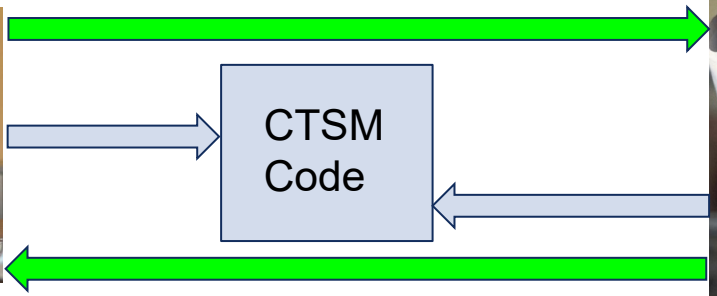
Erik Kluzek, SEWG CESM Workshop 2025 meeting, June 10th

erik@ucar.edu

And CTSM SE Team: Adrianna Foster, Greg Lemieux, Ryan Knox, Sam Levis, Sam Rabin, Will Wieder

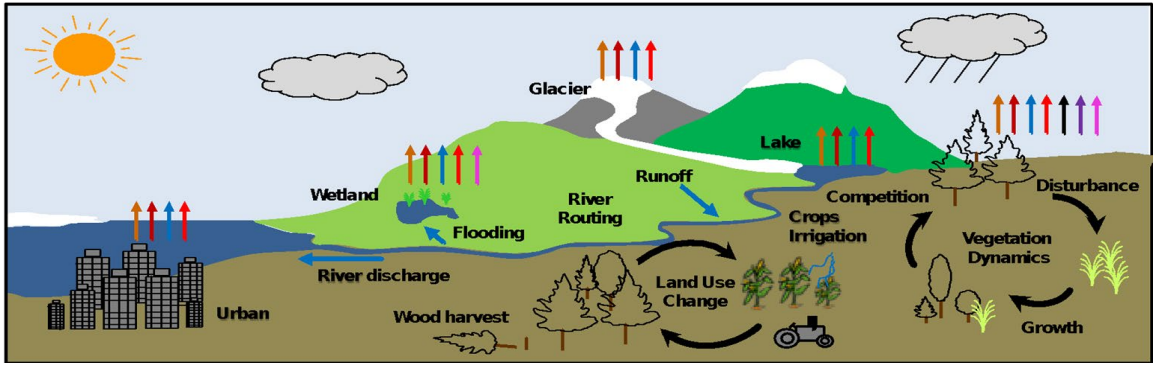
Problem Statement

- ***Scientists contribute a substantial amount of code*** – untrained in RSE practices
- Scientists need to concentrate on their science
- What if we increase collaboration between RSE's and code contributors?
- This is our story of adding lines of collaboration
- ***Goal: Encourage better Software Development practices*** from everyone



Why Should CESM Scientists Care about the Code?

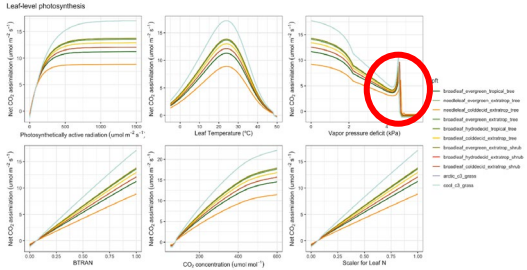
- CESM is Science – expressed in Software
- Therefore if you touch any CESM code – you are an RSE!
- Bad software practices also kill our science
- Clean software and software development practices makes CESM Science:
  - Easier
  - Flexible
  - Robust
  - Verifiable
  - Correct
  - Reproducible



How do we Encourage Scientists to improve Coding Practices?

Help them to see that poor code impacts the science

- When problems come up examine the causes and how to prevent
- Point out the missing practices that could've helped
- Point out the cost of having poor practices and how that's greater than cost of practices



Summary: What Does Work in a Community Model?

Demonstrate costs for bad practices

Demonstrate practices

Educate on practices

Work with people with the highest interests in RSE work

How do we Encourage Scientists to improve Coding Practices?

Educate them on the most important practices

- RSE team actively doing and experts in the practices
- Have education resources available (tutorials, documentation etc.)
- Use metaphors and non-technical terms
- Encourage people that try and provide help
- Education on the practices needs to continue to be done
- Introduce one practice at a time
- Make most important things easy

What Does NOT Work in a Community Model?

- Demanding – “you must do ____”
- Expecting – “you should know ____”
- Shaming – “(condescendingly) I told you so – you should have done ____”
- Technical – “Read this textbook on Software Development Methodology”
- Claiming – “We HAVE to be exactly like the SE Industry”
- Academic – “Read and digest this study on what it shows”
- Overwhelm – “Do ALL of these 10 things at once”
- No relevance – “Do these RSE things – forget about the science”
- Wishing – “Maybe someone will do this on their own”

What Makes Doing RSE Tasks Hard?

What can we learn from the SE industry?

- Debugging is the MOST expensive thing we do
  - Inherently intractable
  - Not possible to estimate
  - Further along in development the more expensive it is
- Brittle, poorly designed, poorly tested code keeps you constantly debugging

CTSM SE Team Problems Debugging Poor Code

- Own example: 83 tasks over 2 years, 50% debugging
- CTSM-SE Team: Sporadic, but 20% ish…
- Identify problems with the code
- Refactor to improve code:
  - Commonly changed
  - Problematic enough

What Practices does the SE Industry Show Helps?

As such SE Industry and Research has found the following practices help:

- Figure out what the software needs to do ***REQUIREMENTS*** (neither too much or too little)
- Spend effort into ***DESIGN*** of the code itself
- Add automated ***TESTING*** WHILE you develop

RSE Maxim to Live By

Untested Code – IS broken (or will get broken)

So…

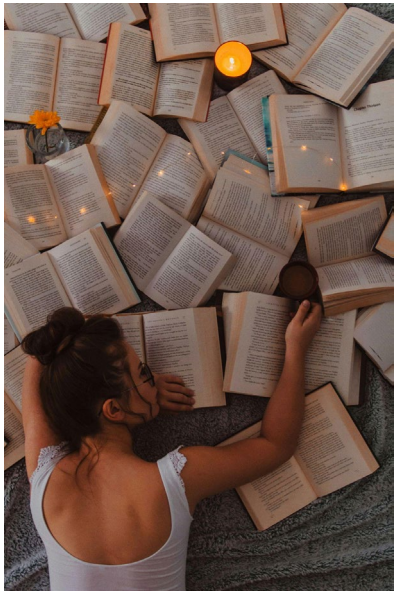
- Don’t add untested code
- Do testing and add testing to test suites
- Continue to run test suites

List of RSE Suggestions

1. Small circle JuJitsu (small cycles)
2. Trim the fat (requirements)
3. Draw the building (design code changes before starting)
4. Preserve success (git version control)
5. Practice vulnerability (openly share code/issues)
6. Trust but verify – test AS you go (Test Driven Development TDD)
7. Improve design as you go (refactoring)

Too Much!

- Each of those 7 practices are important!
- But, it's also too much to expect people to be able absorb
- To see the details go to my NSF-NCAR Software Engineering Assembly (SEA) Improving Scientific Software (ISS) talk from 2025
- So here I'll concentrate on the things we are trying in CTSM in this space…



What are Things we are Trying in CTSM?

1. CTSM Parallel work sessions to mix RSE's and Scientists (also hackathons)
2. Easier to add non-answer changes on our b4b-dev branch
3. Giving CTSM scientists a stable dev version to use with minor version updates
4. With the stable dev version we are free'er to bring in smaller updates on main dev
5. Python unit testing
6. Educating more in PF unit based Fortran unit testing
7. Scientists using Fortran Functional testing
8. Survey with my poster at the workshop

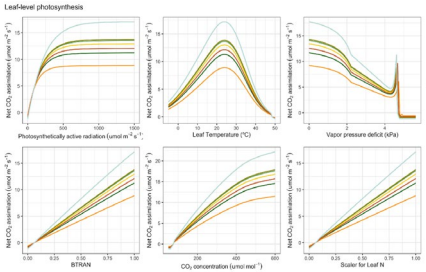
PF Unit Testing for Fortran Code

- We’ve had this in place for a long time
- Taught Keith Oleason to use a few weeks ago
- He was able to figure it out and make use of it
- See’s the point and will do again….

Functional Unit Testing for Science Code

- **Adrianna Foster:**

  - » Unit-testing framework to build drivers for science subroutines
  - » Includes netCDF input and output
  - » Includes python plotting
  - » Teaching this methodology to:
    - grad student: Zhiyi Zhou (Zoey) - CSU
    - postdoc: Xiulin Gao - LB

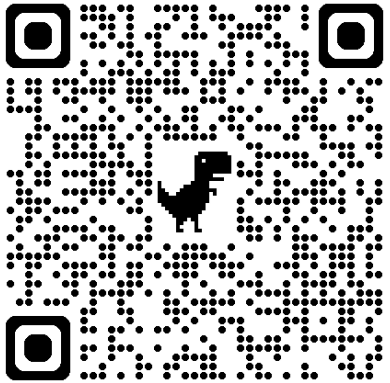


- **Linnia Hawkins:**

  - » Using this framework for faster development of a ML parameterization
  - » Run interactively rather than in job queues
  - » Builds and runs faster
  - » Input and output is simpler

Survey for the Workshop

- Survey on computing familiarity
- On difficult debugging experiences
- And interest in learning more RSE practices
- Please take it
- The QR code is also on my poster



Questions?

- My questions for you:
  - How do we encourage scientists working on code to improve their development practices?
  - How do we measure success?
  - Again, take the survey

Rate of RSE activities for CTSM

- What percentage of CESM should be spent on RSE work? What percent is?
- In TSS in CGD NSF-NCAR 2 of 16 have a title of SE (12%)
- Since 2013 we make 40-60 tags per year (once a week) – increasing this by being more focused
- Added b4b-dev to bring in smaller changes every 2-weeks

- 2024 we opened 190 issues, and closed 261 (3 per tag)
- 2024 we closed 213 Pull Requests (oldest 6 years old)
- We have 43 open Pull Requests in CTSM right now (oldest 6 years old)

“Off-offline” testing

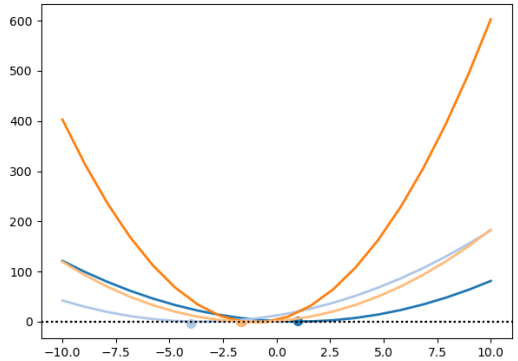
- Test small, isolated pieces of the model
- Validate internal model behavior before running full-scale runs

```python
def plot_quad_and_roots(a_coeff, b_coeff, c_coeff, root1, root2):
    """Plots a set of quadratic formulas (ax**2 + bx + c) and their two roots

    Args:
        a_coeff (float array): set of a coefficients
        b_coeff (float array): set of b coefficients
        c_coeff (float array): set of b coefficients
        root1 (float array): set of first real roots
        root2 (float array): set of second real roots
    """
    num_equations = len(a_coeff)

    plt.figure(figsize=(7, 5))
    x_vals = np.linspace(-10.0, 10.0, num=20)

    colors = get_color_palette(num_equations)
    for i in range(num_equations):
        y_vals = a_coeff[i]*x_vals**2 + b_coeff[i]*x_vals + c_coeff[i]
        plt.plot(x_vals, y_vals, lw=2, color=colors[i])
        plt.scatter(root1[i], root2[i], color=colors[i], s=50)
        plt.axhline(y=0.0, color='k', linestyle='dotted')
```



Easier to add non-answer changes on our b4b-dev branch

- Easier to get in than tags on main-dev
- Tags queued in main-dev are often backed up in a queue
- Doesn’t require a tag and ChangeLog update
- Testing is easier because baselines aren’t stored and only aux_clm required
- We have more people authorized to do this than for tags on main-dev
- Makes it easier to make a simpler, smaller, well confined change that does one thing
- Easier to review these smaller changes
- The b4b-dev branch gets merged to main-dev every 2 weeks

Stable Minor Version for Science

- We are now doing regular minor version updates, second number increment (ctsm5.2.0, ctsm5.3.0, ctsm5.4.0 etc.)
- Each minor version has a “blessed” release version for scientists to use (ctsm5.2.05, ctsm5.3.021, etc.)
- Development release version has more documentation of changes since last one
- Look at “release versions” on CTSM github:
  https://github.com/ESCOMP/CTSM/releases
- Because there is a blessed versions for scientists to use we are now more free to bring in smaller changes and things that might break
- We can bring in small “hotfix” updates that fix a small problem (and we are)

What do some CTSM Scientists Do with Code they Find is Broken?

When you try something in CESM and it's broken for you -- what do you do?
(Caveat – small group, and informal survey)

What Can we Learn from This?

- Most people ask someone else
  - That means we have a community to help
  - This is time consuming for CESM RSE staff
  - This communication usually doesn't result in a fix coming in
- About a quarter ask on forums or create an issue
  - We may want to increase this fraction
- A small number could fix the problem
  - Shows knowledge at least for some in the group
  - But, also shows in general the model is complex enough that this is hard to do
  - Hence improving our testing should help all users
- No one selected "give up and move on"
  - This is good news

Testing leaf -level photosynthesis

```
! calculate canopy gas parameters
call GetCanopyGasParameters(can_air_press, can_o2_pp, veg_tempk, can_tempk,           &
  can_vpress, veg_esat, leaf_bl_resistance, mm_kco2, mm_ko2, co2_compensation_pt,     &
  cf, leaf_bl_conductance, can_vpress_constrained)

! calculate leaf biophysical rates
call LeafLayerBiophysicalRates(par, ft, EDPftvarcon_inst%vcmax25top(ft,1),            &
  param_derived%jmax25top(ft,1), param_derived%kp25top(ft,1), nscaler, veg_tempk,     &
  dayl_fact, can_tempk, can_tempk, btran, vcmax, jmax, co2_rcurve_islope)

! pulled out from model for now
stomatal_int_btran = max(cf/2.E8_r8, EDPftvarcon_inst%stomatal_intercept(ft)*btran)
      You, 7 months ago • start adding other scalers
! calculate leaf-level photosynthesis
call LeafLayerPhotosynthesis(1.0_r8, par, 0.0_r8, 10.0_r8, 0.0_r8, ft, vcmax, jmax,  &
  co2_rcurve_islope, veg_tempk, veg_esat, can_air_press, can_co2_pp, can_o2_pp, btran, &
  stomatal_int_btran, cf, leaf_bl_conductance, can_vpress_constrained, mm_kco2,       &
  mm_ko2, co2_compensation_pt, leaf_maintenance_resp, 999.0_r8, leaf_bl_resistance,   &
  assimilation, stomatal_resistance, leaf_photo, c13disc, test_out)
```

Testing leaf -level photosynthesis