Unit Testing in CESM

Bill Sacks
CESM Software Engineering Group (CSEG)
NCAR / CGD

with substantial contributions from
Sean Santos (U Washington, formerly CSEG)

See also:
https://github.com/NCAR/cesm_unit_test_tutorial
Outline

• What is a unit test

• Examples of unit testing in CESM

• CESM unit testing tutorial & other references
Outline

• What is a unit test

• Examples of unit testing in CESM

• CESM unit testing tutorial & other references
What is a Unit Test?

A fast, self-verifying test of a small piece of code
What is a Unit Test?

A fast, self-verifying test of a small piece of code

Order milliseconds
What is a Unit Test?

A fast, self-verifying test of a small piece of code

- Order milliseconds
- Instant PASS/FAIL
- No manual analysis
What is a Unit Test?

A fast, self-verifying test of a small piece of code

- Order milliseconds
- Instant PASS/FAIL
- No manual analysis
- Order 10 - 100 lines
  Single function or small module
What is a Unit Test?

A fast, self-verifying test of a small piece of code

- Order milliseconds
- Instant PASS/FAIL
- No manual analysis
- Order 10 - 100 lines
- Single function or small module

@Test
subroutine windCompactionFactor_topLayer_hugeWind_returns_maxFactor(this)
  ! If wind is essentially infinite, the return value should be MAX_FACTOR
  class(TestSnowHydrology), intent(inout) :: this
  real(r8) :: windcomp

  windcomp = WindCompactionFactor( &
      forc_wind = huge(1._r8), &
      layer     = STANDARD_TOP_LAYER, &
      snl       = STANDARD_SNL)

  @assertEqual(MAX_FACTOR, windcomp, tolerance=tol)
end subroutine windCompactionFactor_topLayer_hugeWind_returns_maxFactor
What are the Benefits of Unit Tests?
What are the Benefits of Unit Tests?

- Ensure code remains correct as it is modified
  - Complement system-level regression tests in this respect
What are the Benefits of Unit Tests?

• Ensure code remains correct as it is modified
  ▸ Complement system-level regression tests in this respect

• Ensure new code is correct
What are the Benefits of Unit Tests?

• Ensure code remains correct as it is modified
  ▸ Complement system-level regression tests in this respect

• Ensure new code is correct

• Tests guiding development
What are the Benefits of Unit Tests?

• Ensure code remains correct as it is modified
  ▸ Complement system-level regression tests in this respect

• Ensure new code is correct

• Tests guiding development

• Tests as documentation
What are the Benefits of Unit Tests?

• Ensure code remains correct as it is modified
  ▸ Complement system-level regression tests in this respect

• Ensure new code is correct

• Tests guiding development

• Tests as documentation

• Support development on your desktop machine
Outline

• What is a unit test

• Examples of unit testing in CESM

• CESM unit testing tutorial & other references
function WindCompactionFactor(forc_wind, layer, snl) result(windcomp)
! Computes compaction enhancement factor of snowpack due to wind.
!
! Parameterization comes from Glen Liston et al., Journal of Glaciology, Vol. 53, No. 181, 2007 (equation 18)
!
! For forc_wind >= 5 m/s, parameterization varies between 5 (forc_wind = 5 m/s) and 20 (forc_wind large); for forc_wind < 5 m/s, returns 1.

real(r8) :: windcomp ! function result
real(r8), intent(in) :: forc_wind ! atmospheric wind speed (m/s)
integer, intent(in) :: layer ! current snow layer index (between snl+1 [top] and 0 [bottom])
integer, intent(in) :: snl ! NEGATIVE number of snow layers

! Only apply wind compaction to top snow layer
if (layer == snl+1) then
  if (forc_wind >= MIN_WIND) then
    windcomp = MIN_FACTOR + MAX_ADDITIONAL_FACTOR * &
      (1.0_r8 - exp(-PROGRESSION_FACTOR * (forc_wind - MIN_WIND)))
  else
    windcomp = 1.0_r8
  end if
else
  windcomp = 1.0_r8
end if
end function WindCompactionFactor
Example: Test of "Standard" Case

components/clm/src/biogeophys/test/SnowHydrology_test/
test_SnowHydrology_windCompactionFactor.pf

! limit of factor for very strong winds
real(r8), parameter :: MAX_FACTOR = MIN_FACTOR + MAX_ADDITIONAL_FACTOR

! Using these "standard" constants should result in a moderate wind compaction factor
integer , parameter :: STANDARD_SNL       = -4                      ! negative number of snow layers
integer , parameter :: STANDARD_TOP_LAYER = STANDARD_SNL + 1
real(r8) , parameter :: STANDARD_WIND      = MIN_WIND * 2._r8 ! moderate wind speed (m/s)

real(r8), parameter :: tol = 1.e-13 ! tolerance for error checks

@Test
subroutine windCompactionFactor_topLayer_moderateWind(this)
  ! Test the "standard" inputs
  class(TestSnowHydrology), intent(inout) :: this
  real(r8) :: windcomp

  windcomp = WindCompactionFactor(
    & forc_wind = STANDARD_WIND, &
    & layer     = STANDARD_TOP_LAYER, &
    & snl       = STANDARD_SNL)

  @assertGreaterThan(windcomp, MIN_FACTOR)
  @assertLessThan(windcomp, MAX_FACTOR)
end subroutine windCompactionFactor_topLayer_moderateWind
Example: Tests of Edge Cases

```fortran
@Test
subroutine windCompactionFactor_topLayer_minWind_returns_minFactor(this)
  ! If wind is at the minimum threshold value, the return value should be MIN_FACTOR
  class(TestSnowHydrology), intent(inout) :: this
  real(r8) :: windcomp

  windcomp = WindCompactionFactor( &
      forc_wind = MIN_WIND, &
      layer     = STANDARD_TOP_LAYER, &
      snl       = STANDARD_SNL)

  @assertEqual(MIN_FACTOR, windcomp)
end subroutine windCompactionFactor_topLayer_minWind_returns_minFactor

@Test
subroutine windCompactionFactor_topLayer_hugeWind_returns_maxFactor(this)
  ! If wind is essentially infinite, the return value should be MAX_FACTOR
  class(TestSnowHydrology), intent(inout) :: this
  real(r8) :: windcomp

  windcomp = WindCompactionFactor( &
      forc_wind = huge(1._r8), &
      layer     = STANDARD_TOP_LAYER, &
      snl       = STANDARD_SNL)

  @assertEqual(MAX_FACTOR, windcomp, tolerance=tol)
end subroutine windCompactionFactor_topLayer_hugeWind_returns_maxFactor
```
Running tests...
Test project /Users/sacks/cesm_code/clm_trunk_withMods/components/clm/src/build
Start  1: unittestArray
1/23 Test  #1: unittestArray ....................   Passed    0.01 sec
Start  2: unittestSubgrid
2/23 Test  #2: unittestSubgrid ..................   Passed    0.01 sec
Start  3: unittestFilterBuilder
3/23 Test  #3: unittestFilterBuilder ............   Passed    0.01 sec
Start  4: clm_time_manager
4/23 Test  #4: clm_time_manager ...................   Passed    0.01 sec
Start  5: daylength
5/23 Test  #5: daylength .........................   Passed    0.01 sec
Start  6: irrigation
6/23 Test  #6: irrigation ..........................   Passed    0.01 sec
Start  7: humanstress
7/23 Test  #7: humanstress .......................   Passed    0.01 sec
Start  8: SnowHydrology
8/23 Test  #8: SnowHydrology .....................   Passed    0.01 sec
Start  9: acspinup
9/23 Test  #9: acspinup ............................   Passed    0.02 sec

[More output cut]

23/23 Test #23: initInterpMultilevel .............   Passed    0.01 sec

100% tests passed, 0 tests failed out of 23

Total Test time (real) =   0.25 sec
windcomp = MIN_FACTOR + MAX_ADDITIONAL_FACTOR * &
(1.0_r8+exp(-PROGRESSION_FACTOR * (forc_wind - MIN_WIND)))

Running tests...

[More output cut]

8/23 Test #8: SnowHydrology .........................***Failed Error regular expression
found in output. Regex=[FAILURES!!!] 0.01 sec
...F.F...
Time: 0.000 seconds

Failure in:
test_SnowHydrology_windCompactionFactor_suite.windCompactionFactor_topLayer_mode
Location: [test_SnowHydrology_windCompactionFactor.pf:57]
expected +25.51819 to be less than: +20.00000.

Failure in:
test_SnowHydrology_windCompactionFactor_suite.windCompactionFactor_topLayer_minW
Location: [test_SnowHydrology_windCompactionFactor.pf:72]
expected +5.000000 but found: +35.00000; difference: |+30.00000| >
tolerance:+0.000000.

FAILURES!!!
Tests run: 7, Failures: 2, Errors: 0
ERROR STOP *** Encountered 1 or more failures/errors during testing. ***

[More output cut]

96% tests passed, 1 tests failed out of 23

Total Test time (real) = 0.22 sec

The following tests FAILED:
8 - SnowHydrology (Failed)
Errors while running CTest
make[1]: *** [test] Error 8
make: *** [test] Error 2
module test_patch_state_updater

! Tests of dynPatchStateUpdaterMod

use pfunit_mod
use dynPatchStateUpdaterMod
use unittestSubgridMod
use unittestSimpleSubgridSetupsMod
! [and others]

implicit none

@TestCase
type, extends(TestCase) :: TestPSUpdater
  ! filter
  integer :: numf
  integer, allocatable :: filter(:)

contains
  procedure :: setUp
  procedure :: tearDown
  procedure :: do_all_setup_with_n_vegetated_patches
end type TestPSUpdater

real(r8), parameter :: tol = 1.e-13_r8
A More Complex Example

contains

! ===================================================================
! Test helpers
! ===================================================================

subroutine setUp(this)
    class(TestPSUpdater), intent(inout) :: this
end subroutine setUp

subroutine tearDown(this)
    class(TestPSUpdater), intent(inout) :: this
    call unittest_subgrid_teardown()
end subroutine tearDown
A More Complex Example

subroutine do_all_setup_with_n_vegetated_patches(this, ps_updater, &
pwtcol_old, pwtcol_new, pft_types)
  class(TestPSUpdater), intent(inout) :: this
  type(patch_state_updater_type), intent(out) :: ps_updater
  real(r8), intent(in) :: pwtcol_old(begp:)
  real(r8), intent(in) :: pwtcol_new(begp:)
  integer, intent(in), optional :: pft_types(:)

  integer :: p

  call setup_n_veg_patches(pwtcol_old, pft_types)
  call filter_from_range(bounds%begp, bounds%endp, &
    this%numf, this%filter)
  ps_updater = patch_state_updater_type(bounds)
  call ps_updater%set_old_weights(bounds)
  do p = bounds%begp, bounds%endp
    patch%wtcol(p) = pwtcol_new(p)
  end do
  call compute_higher_order_weights(bounds)
  call ps_updater%set_new_weights(bounds)
end subroutine do_all_setup_with_n_vegetated_patches
@Test
subroutine areaIncreases(this)
! if area increases from non-zero, then value is decreased
! appropriately; there should be no change in flux
class(TestPSUpdater), intent(inout) :: this
! The second patch is the patch of interest
real(r8), parameter :: pwtcol_old(3) = [0.3_r8, 0.2_r8, 0.5_r8]
real(r8), parameter :: pwtcol_new(3) = [0.2_r8, 0.3_r8, 0.5_r8]
type(patch_state_updater_type) :: ps_updater
real(r8) :: var(3)
real(r8) :: flux(3)

! Setup
call this%do_all_setup_with_n_vegetated_patches(ps_updater, &
    pwtcol_old, pwtcol_new)
var = [2._r8, 3._r8, 4._r8]
flux = [12._r8, 13._r8, 14._r8]

! Exercise
call ps_updater%update_patch_state(bounds, &
    this%numf, this%filter, var, flux)

! Verify
@assertEqual(2._r8, var(2))
! same as starting flux:
@assertEqual(13._r8, flux(2), tolerance=tol)
end subroutine areaIncreases
The Scientific Software Testing Challenge

We usually don't know the right answer ahead of time

Possible solutions

• Break code into smaller pieces
  ▷ Don't try to test that 1000-line monster subroutine all at once

• Test boundary conditions where it's easier to determine the right answer

• Work through one or two cases by hand
Unit Test Status in CESM

At least partial coverage of 45 Fortran modules

- share code: 9 modules
  - Low-level infrastructure
  - Shared science routines

- coupler: 6 modules
  - Mainly mapping functions between land and glacier

- CAM: 3 modules
  - Some lower-level infrastructure

- CLM: 27 modules
  - Some low-level infrastructure code
  - Much of the dynamic landunits infrastructure
  - Some science code, including complex tests of irrigation
Outline

• What is a unit test
• Examples of unit testing in CESM
• CESM unit testing tutorial & other references
Tutorial on Building & Running Unit Tests in CESM

https://github.com/NCAR/cesm_unit_test_tutorial/

• Instructions for CLM, CAM, csm_share and driver_cpl

• How to build & run existing unit tests
  ▸ On yellowstone
  ▸ On your own machine

• Where to put new unit tests

• How to set up the build for your new unit tests

• Tips on writing good unit tests

• Documentation of commonly-used pFUnit assertions & other features

• Tips on unit test teardown
More Documentation and Examples of Unit Testing in CESM

- `cime/tools/unit_testing`
  - README
  - Examples

- README files for how to run tests
  - `components/clm/src/README.unit_testing`
  - `cime/README.unit_testing`

- See other examples:
  
  `find . -name '*pf'`

- Unit test build: See various CMakeLists.txt files

- I'm happy to help: sacks@ucar.edu
Good Unit Testing Books

Good introductions to effective unit testing

Good references for writing unit tests in more complex cases
General Unit Testing Guidelines
Unit Test Structure

• A given unit test should test a single, well-defined condition
  ▶ Easier to pinpoint a problem when a test fails
  ▶ Tests as documentation

• Four distinct pieces
  ▶ Setup (optional)
  ▶ Exercise
  ▶ Verify
  ▶ Teardown (optional)
    - Pointer deallocation, resetting global variables, etc.
    - Should be done in a teardown method
Good Unit Tests are FIRST

• Fast (milliseconds or less)
  ▸ Avoid file I/O, test with small arrays, etc.

• Independent
  ▸ e.g., test Y shouldn’t depend on global variables set by test X

• Repeatable
  ▸ e.g., don’t generate random numbers in your tests

• Self-verifying
  ▸ Should generate an automatic pass/fail

• Timely
  ▸ Tests written before production code (test-driven development) or immediately afterwards