Software Testing: Introduction

Gregory R. Watson (he/him)
Oak Ridge National Laboratory

Better Scientific Software tutorial @ SC21

Contributors: Anshu Dubey (ANL), Rinku Gupta (ANL), Alicia Klinvex (SNL), Mark C. Miller (LLNL), Jared O'Neal (ANL), Patricia Grubel (LANL), David M. Rogers (ORNL), Gregory R. Watson (ORNL)

LA-UR-21-25675

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Patricia A. Grubel, Rinku K. Gupta, and Gregory R. Watson, Better Scientific Software tutorial, in the International Conference for High-Performance Computing, Networking, Storage, and Analysis (SC21), St. Louis, MO, USA and online, 2021. DOI: 10.6084/m9.figshare.16556628**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial…

Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
- This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.
- This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No.89233218CNA000001
- This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Software Testing - Outline

Testing Introduction

- Context
- Challenges
- Simple Examples

Testing Walkthrough

- Walk Through Testing Example

Audiences for this presentation

- New to testing / beginning development on a new project
  - Helpful terminology and starting points for how to get going
- Working with a legacy project that needs testing
  - Integrating a test framework and writing tests
- Improving testing practices on an existing project
  - Different testing strategies and how to approach test driven development

Testing types and terminology

- There are many different testing techniques and associated terminology
  - We're focusing on "dynamic testing", i.e. testing by code execution
- Functional testing is testing all the components systematically against requirements or specifications
  - Common types: unit, integration, system, acceptance, regression, etc.
- Non-functional testing tests how the program operates, rather than a specific set of behaviors
  - Common types: performance, security, usability, etc.

Example types of testing

- Unit testing – verify the execution of a single routine
- Integration testing – verify that modules or components execute correctly
- System testing – verify that the program operates correctly as a whole
- Regression testing – comparison with previous output to determine if unintended changes have been introduced
- Acceptance testing – verify that the program meets customer requirements or not

Note that this is neither a complete nor prescriptive list of testing types. There are many other types of tests. Sometimes it can be beneficial to combine testing strategies, such as testing the interoperability of modules and components at the system level.

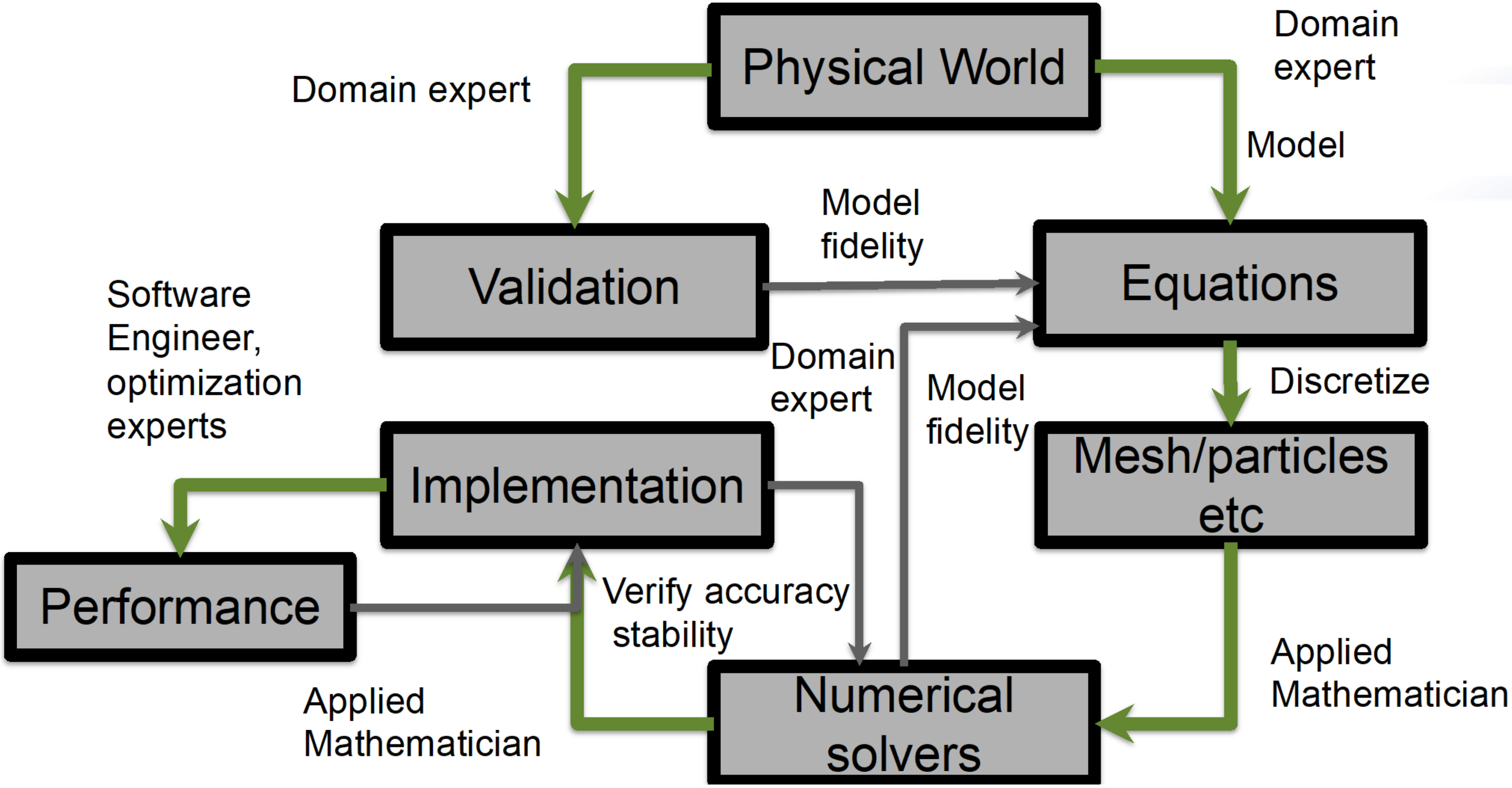
What about Verification and Validation?

- Scientific computing and software engineering use different definitions

	Scientific computing	Software engineering
Verification	Confirms the mathematical accuracy and stability of a numerical solution in addition to specifications.	Confirms that the software conforms to its specifications (i.e. requirements.)
Validation	Confirms the physical accuracy of a given model by comparing against experimental data.	Confirms that the software actually meets the customer's needs.

- Validation in scientific computing requires a comparison to the experimental data, whereas in software engineering it is based on customer needs
- Also, for a real problem, there is typically no way to check for correct output given some inputs. Validation is still required however, so an indirect method must be used.

Testing within the software development lifecycle



Testing within the software development lifecycle

- When should functional tests be provided?
- Ideally before the code is written
  - Also known as test driven development (TDD)
  - Tests then become the specification for the program
- This approach also ensures that thought is given to what it means for the program to be correct, rather than just what the program should do
- Requires:
  - Care in writing tests
  - Frequent running of tests (see our Continuous Integration module)
  - Wide adoption by development team

Steps for test driven development

- Write a single test[1] describing an aspect of the program
- Run the test, which should fail because the feature does not exist
- Write just enough code to make the test pass
- Refactor the code
- Repeat, creating new tests as new functionality is added

[1]In numerical methods there are times when a single test may not suffice

Challenges

- Exploratory Software
  - Implies one does not know the outcome
  - Still determining where model is valid
  - Validation from domain experts feeds back into design
- Legacy Codes
  - Original verification has been lost in the mists of time.
  - Assumptions, conditions, interactions unknown: “Bad code or necessary evil?”
- Releasing Codes
  - Code review to check scope of problem, solution, and documentation.
  - Verification before product release is a cost-effective way to prevent defects from getting through.

Python Example

```
$ pip install pyscaffold
$ pip install tox
$ putup autoQCT
$ cd autoQCT # tests in tests/ subdir.
$ tox
```

```
default run-test: commands[0] | pytest
======================= test session starts ========================
platform darwin -- Python 3.9.0, pytest-6.2.2, py-1.10.0, pluggy-0.13.1 -- plugins: cov-2.11.1
collected 2 items

tests/test_skeleton.py::test_fib PASSED                    [ 50%]
tests/test_skeleton.py::test_main PASSED                   [100%]

---------- coverage: platform darwin, python 3.9.0-final-0 -----------
Name                      Stmts   Miss Branch BrPart  Cover   Missing
---------------------------------------------------------------------
src/autoqct/__init__.py       6      0      0      0   100%
src/autoqct/skeleton.py      32      1      2      0    97%   135
---------------------------------------------------------------------
TOTAL                        38      1      2      0    98%

======================== 2 passed in 0.07s =========================
  default: commands succeeded
  congratulations :)
```

PYSCAFFOLD

pyscaffold.org

CMake Example

```
$ cat >CMakeLists.txt <<.
cmake_minimum_required(VERSION 3.8)
project( blank )
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
include(blt/SetupBLT.cmake)
.
$ git clone https://github.com/LLNL/blt/
$ mkdir build && cd build
$ cmake ..
$ make –j
```

```
...
[100%] Linking CXX executable ../../../tests/blt_gtest_smoke
[100%] Built target blt_gtest_smoke
$ make test
Running tests...
Test project /Users/99r/work/autoQCT/blank_project/build
    Start 1: blt_gtest_smoke
1/1 Test #1: blt_gtest_smoke ..................   Passed    0.46 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.46 sec
```

llnl-blt.readthedocs.io

EXASCALE COMPUTING PROJECT

Going Further

- C, C++, Fortran
  - Running and Reporting Tests: ctest / cdash / pFUnit / FlashTest
  - Code Coverage: gcov / lcov (C, C++, Fortran)
  - Static Analysis: clang-tidy (only C, C++)
- Python
  - Running and Reporting Tests: pytest / unittest / nose
  - Code Coverage: pytest-cov
  - Static Source Code Analysis: pylint / flake8

How do we determine what other tests are needed?

Code coverage tools

- Expose parts of the code that aren't being tested
  - gcov - standard utility with the GNU compiler collection suite (we will use it in the next few slides)
  - Compile/link with –coverage & turn off optimization
  - Counts the number of times each statement is executed
  - Necessary for testing, but not sufficient
- gcov also works for C and Fortran
  - Other tools exist for other languages
  - JCov for Java
  - Coverage.py for python
  - Devel::Cover for perl
  - profile for MATLAB
- Lcov
  - a graphical front-end for gcov
  - available at http://ltp.sourceforge.net/coverage/lcov.php
  - Codecov.io in CI module
- Hosted servers (e.g. coveralls, codecov)
- graphical visualization of results
- push results to server through continuous integration server

Checking coverage Example

- Example of heat equation
  - Add -coverage as shown below to Makefile
  - Run ./heat runame="ftcs_results"
  - Run gcov heat.C
  - Examine heat.C.gcov

- A dash indicates non-executable line
- A number indicated the times the line was called
- ##### indicates line wasn't exercised

```
HDR = Double.H
SRC = heat.C utils.C args.C exact.C ftcs.C upwind15.C crankn.C
OBJ = $(SRC:.C=.o)
GCOV = $(SRC:.C=.C.gcov) $(SRC:.C=.gcda) $(SRC:.C=.gcno) $(HDR:.
H=.H.gcov)
EXE = heat

# Implicit rule for object files
%.o : %.C
	$(CXX) -c -coverage $(CXXFLAGS) $(CPPFLAGS) $< -o $@

# Linking the final heat app
heat: $(OBJ)
	$(CXX) -coverage -o heat $(OBJ) $(LDFLAGS) -lm
```

```
    -:  143:static bool
  500:  144:update_solution()
    -:  145:{
  500:  146:    if (!strcmp(alg, "ftcs"))
  500:  147:        return update_solution_ftcs(Nx, curr, last, alpha, dx, dt, bc0, bc1);
#####:  148:    else if (!strcmp(alg, "upwind15"))
#####:  149:        return update_solution_upwind15(Nx, curr, last, alpha, dx, dt, bc0, bc1);
#####:  150:    else if (!strcmp(alg, "crankn"))
#####:  151:        return update_solution_crankn(Nx, curr, last, cn_Amat, bc0, bc1);
#####:  152:    return false;
  500:  153:}
    -:  154:
    -:  155:static Double
  500:  156:update_output_files(int ti)
    -:  157:{
  500:  158:    Double change;
    -:  159:
  500:  160:    if (ti>0 && save)
    -:  161:    {
#####:  162:        compute_exact_solution(Nx, exact, dx, ic, alpha, ti*dt, bc0, bc1);
#####:  163:        if (savi && ti%savi==0)
#####:  164:            write_array(ti, Nx, dx, exact);
#####:  165:    }
```

Graphical View of Gcov Output and Tutorials for Code Coverage

Coverage Summary

SOURCE FILES ON BUILD 45

LIST 2 CHANGED 0 SOURCE CHANGED 0 COVERAGE CHANGED 0

COVERAGE	Δ	FILE	LINES	RELEVANT	COVERED
74.39		src/functions/linear_fcn_class.f90	301	82	61
100.0		src/general/modulo_mod.f90	52	3	3

Line-by-line details

```
    ! Error distribution same for all x values
    delta = S*Sxx - Sx*Sx
    if (delta == 0.0_wp) then
        ERRORMSG("Cannot do linear least-sqrs.  Divide by zero.")
        stop
    end if
    delta_inv = 1.0_wp / delta
```

Online tutorial - https://github.com/amklinv/morpheus

Other example - https://github.com/jrdoneal/infrastructure

Hello Numerical World Example (heat equation)

https://github.com/bssw-tutorial/hello-numerical-world

```
$ wc *.C
   125    494   4161 args.C      # parse arguments
   220    718   5667 heat.C      # main() – stores all vars
   151    498   3888 utils.C     # l2_norm, write, copy, init
    26    119    820 ftcs.C      # standard, centered stencil
    27    123    833 upwind15.C  # alternate integration schemes
    94    344   2134 crankn.C
    43    190   1299 exact.C     # comparison solution
```

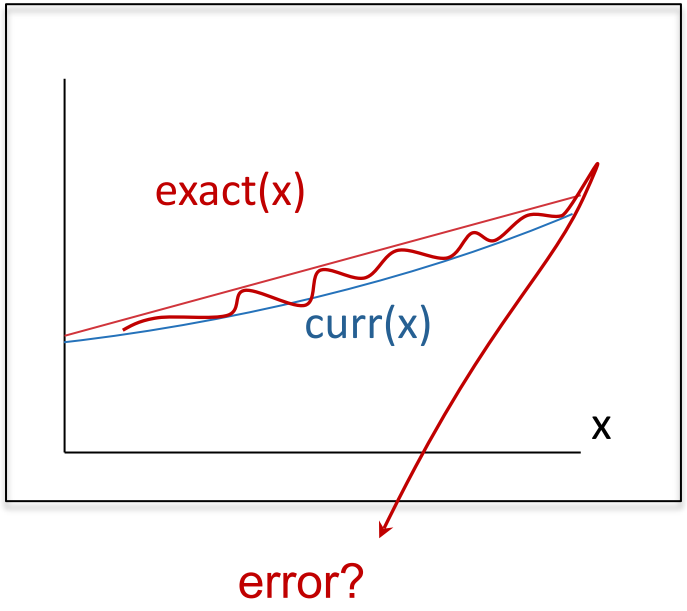
- Lots of setup code – prepares problem for kernel calls
- Isolated, swappable kernel calls
  - Imagine adding kernels to larger, multi-physics application.
- How do we add new kernels using test driven development?

Build and run the code

```
$ cmake .
$ make
$ ./heat alg=ftcs outi=0 maxt=-5e-8 ic="rand(0,0.2,2)"
      runame="heat_results"
      alpha=0.2
      lenx=1
      dx=0.1
      dt=0.004
      maxt=-5e-08
      bc0=0
      bc1=1
      ic="rand(0,0.2,2)"
      alg="ftcs"
      …
Stopped after 001490 iterations for threshold 2.46636e-15
```

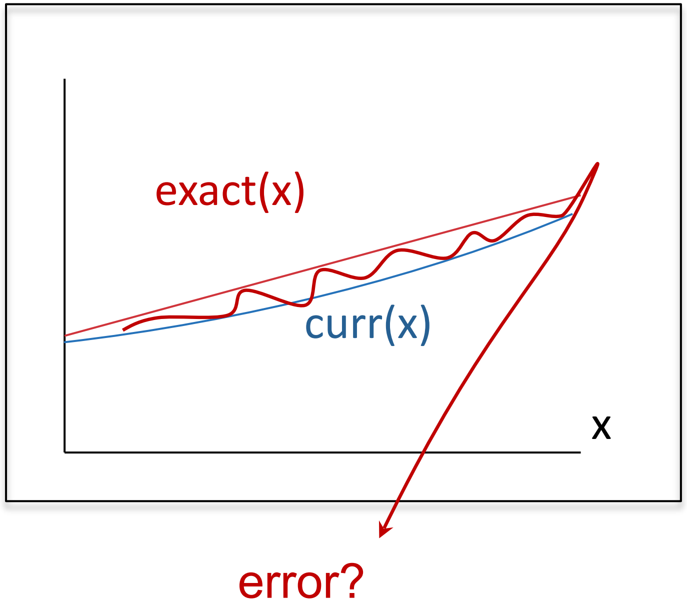
Add a new kernel

- Need to add test first
- What to test?
  - ensure arguments are correct, bad cases detected, etc.
  - steady-state (should be straight line)
    - external script can test file write() as well
  - solution time-dependence vs. reference
    - $(d/dx)^2 \sin(ax) = -a^2 \sin(ax)$
  - test multiple precisions
    - combinatorial problems – listing tests in for() or matrix...



check.sh will check for steady state

```
$ sh check.sh ./heat ftcs
 runame="check_ftcs"
    alpha=0.2
    lenx=1
    dx=0.1
    dt=0.004
    maxt=-5e-08
    bc0=0
    bc1=1
    …
Stopped after 001490 iterations for threshold 2.46636e-15
Error = 0
```



steady-state test
(should be straight line)

Create our tests before we write the code

```
$ mkdir tests
$ cp check.sh tests/check.sh
$ cd tests
```

- Create CMakeLists.txt
- Add test for new kernel
- Optionally add test for existing kernel

```
enable_testing()

add_test(NAME ftcs COMMAND check.sh $<TARGET_FILE:heat> ftcs)
add_test(NAME upwind15 COMMAND check.sh $<TARGET_FILE:heat> upwind15)
```

https://cmake.org/cmake/help/latest/command/add_test.html

Re-run cmake to enable tests

```
$ cd ..
$ cmake -DBUILD_TESTS=ON .
$ cd tests
```

- Run tests

```
$ ctest
Test project /home/tutorial/hello-numerical-world/tests
    Start 1: ftcs
1/2 Test #1: ftcs .............................   Passed    0.23 sec
    Start 2: upwind15
2/2 Test #2: upwind15 .........................***Failed    0.01 sec

50% tests passed, 1 tests failed out of 2

Total Test time (real) =   0.25 sec

The following tests FAILED:
              2 - upwind15 (Failed)
Errors while running CTest
```

Add new kernel to make test succeed

heat.C

- Add prototype

```
extern bool
update_solution_upwind15(int n,
    Double *curr, Double const *last,
    Double alpha, Double dx, Double dt,
    Double bc_0, Double bc_1);
```

- Modify assertion

```
    assert(strncmp(alg, "ftcs", 4)==0 || strncmp(alg, "upwind15", 8)==0);
```

- Call kernel

```
    if (!strcmp(alg, "ftcs"))
        return update_solution_ftcs(Nx, curr, last, alpha, dx, dt, bc0, bc1);
    else if (!strcmp(alg, "upwind15"))
        return update_solution_upwind15(Nx, curr, last, alpha, dx, dt, bc0, bc1);
```

Add new kernel to build

CMakeLists.txt

```
add_executable(heat args.C
                exact.C
                 heat.C
                 upwind15.C  <<< Add new kernel
                 ftcs.C
                 utils.C)
```

- Re-build executable

```
$ make
-- Configuring done
-- Generating done
-- Build files have been written to: /home/tutorial/hello-numerical-world
Scanning dependencies of target heat
[ 14%] Building CXX object CMakeFiles/heat.dir/heat.C.o
[ 28%] Building CXX object CMakeFiles/heat.dir/upwind15.C.o
[ 42%] Linking CXX executable heat
[100%] Built target heat
```

Re-run tests

```
$ cd tests
$ ctest
Test project /home/tutorial/hello-numerical-world/tests
    Start 1: ftcs
1/2 Test #1: ftcs .............................   Passed    0.23 sec
    Start 2: upwind15
2/2 Test #2: upwind15 .........................   Passed    0.03 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   0.13 sec
```

- Succeeded!
- This is all there is to start to use test driven development

Going Further

- Add another kernel
  - crankn.C
  - Note: requires an extra initialization step!
- Reproduce these testing strategies on another repository
  - https://github.com/frobnitzem/simple-heateq (same problem, different design)
- Brainstorm some simple tests you could add to your own project
  - checks you've run manually
  - difficult-to-setup and reproduce cases that could be automated
- Add some "blank tests" to your project
  - reduces the barrier to increased testing
  - What would make reporting on your build / run status better/simpler/more accessible?

Summary

- A productive software team is always checking their work.
  - Take time to recognize these checks and harden them into “real,” repeatable tests.
- Test layout should mirror the logical structure of your code.
  - Test each module, being aware of module to module dependencies.
- Different challenges are associated with exploratory, legacy, and composable codes
  - Adapt your strategy to fit your situation.
  - Eventually you will want to be able to verify all components in a code release.
- Don’t get distracted by all the technologies out there – focus on exercising your code.
  - Scaffolding projects can help with mechanics.

Resources

- Oberkampf, W., & Roy, C. (2010). Verification and Validation in Scientific Computing. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511760396
- Michael Feathers. 2004. Working Effectively with Legacy Code. Prentice Hall PTR, USA. ISBN: 9780131177055