Software Testing: Introduction

David M. Rogers
Oak Ridge National Laboratory

Software Productivity and Sustainability track, ATPESC 2021

Contributors: Anshu Dubey (ANL), Rinku Gupta (ANL), Alicia Klinvex (SNL), Mark C. Miller (LLNL), Jared O'Neal (ANL), Patricia Grubel (LANL)

LA-UR-21-25675

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Rinku K. Gupta, and David M. Rogers, Software Productivity and Sustainability track, in Argonne Training Program on Extreme-Scale Computing (ATPESC), online, 2021. DOI: 10.6084/m9.figshare.15130590**
- 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

- Development context for testing
- Challenges
- Toy Example

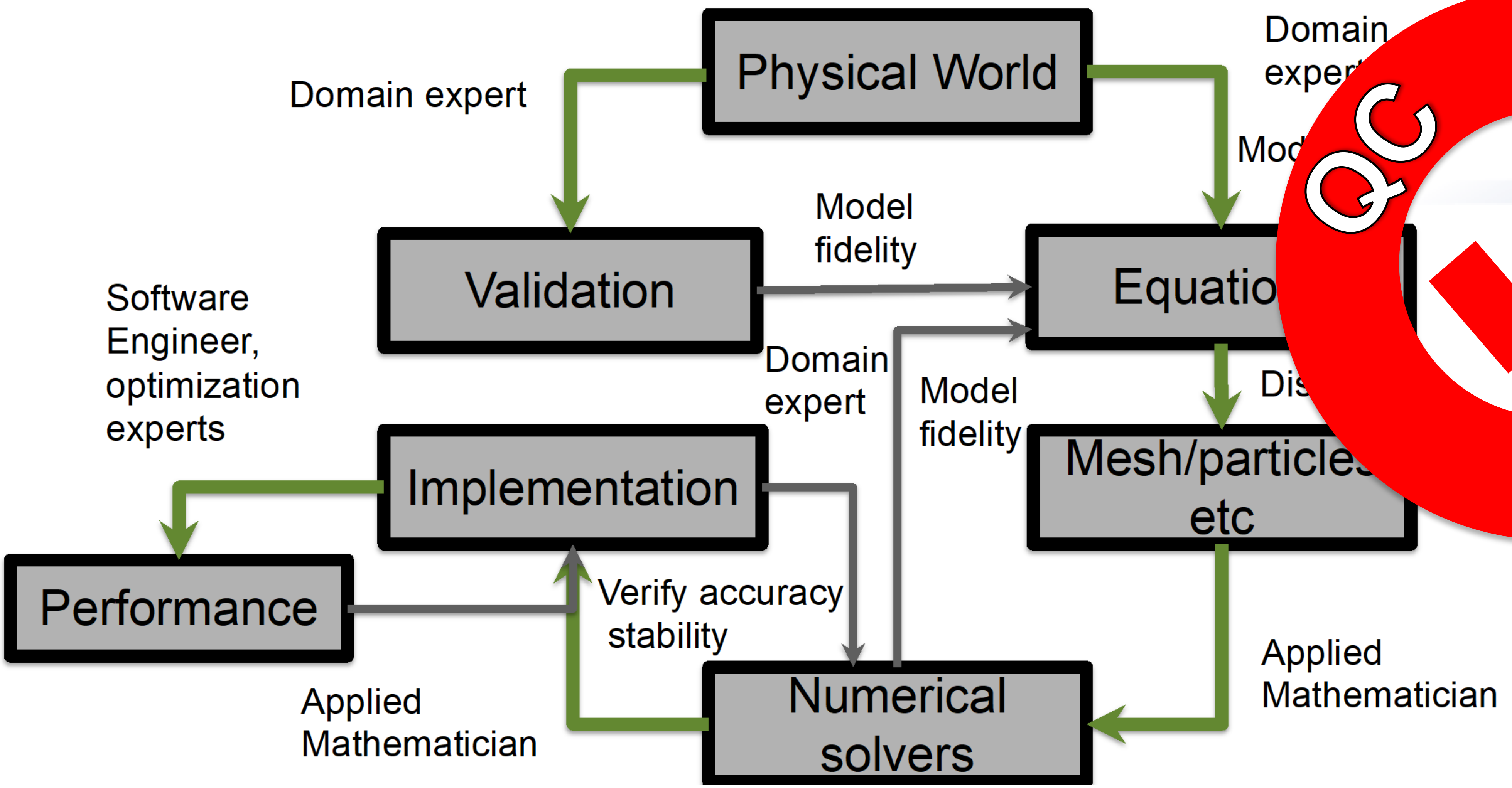
Testing Walkthrough

- Walk Through Testing Example

Advanced Testing

- Guidelines for developing a testing & validation plan
- Production Examples
  - Testing a legacy Fortran code
  - Designing tests alongside code development
- Conclusions: Testing within a team context

Testing within the software development lifecycle



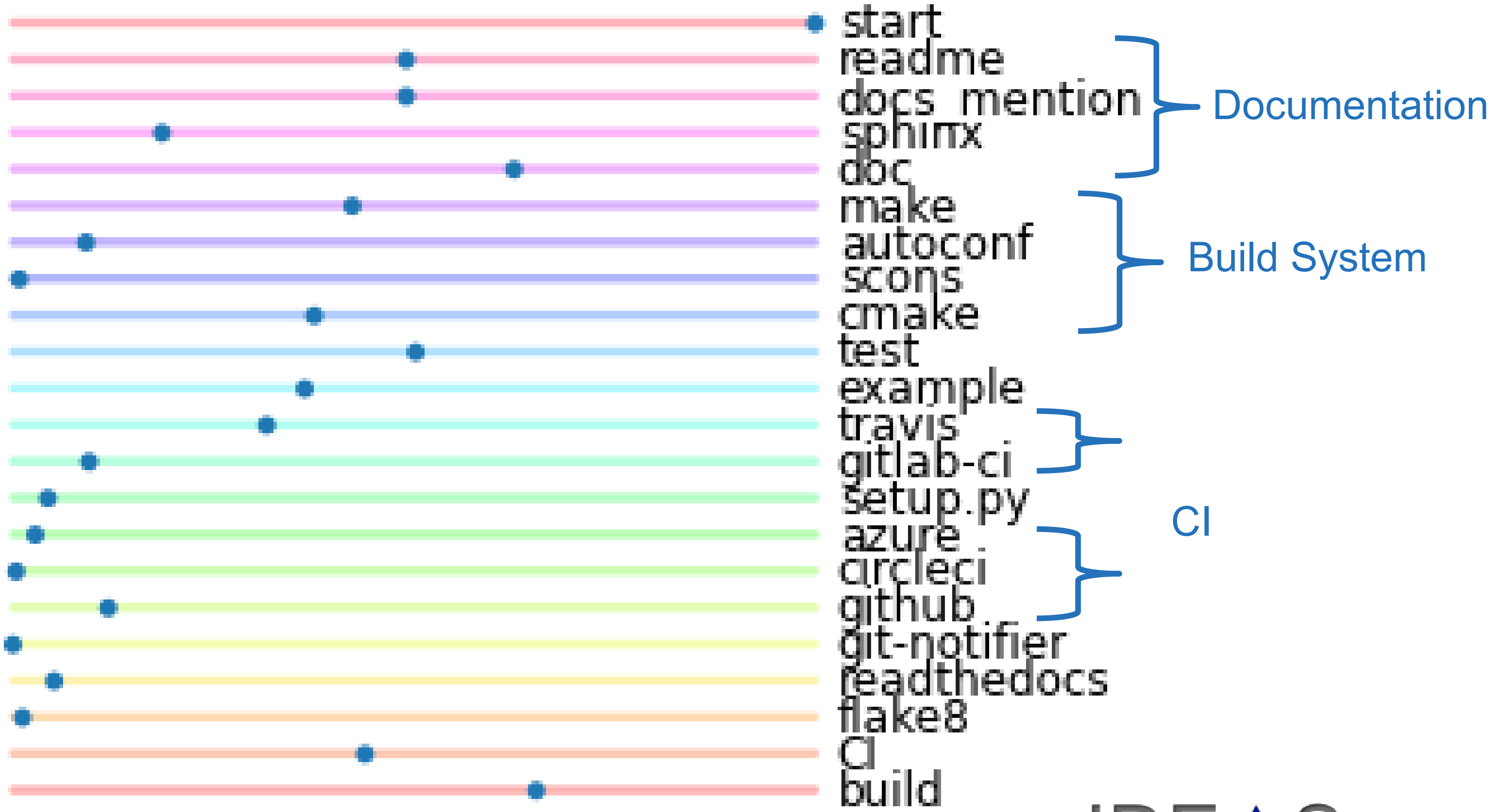
IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Testing within the software development lifecycle

- During initial code development
  - Accuracy and stability
  - Matching the algorithm to the model
  - Interoperability of algorithms
- In later stages
  - Adding new major capabilities
  - Modifying existing capabilities
  - Ongoing maintenance
  - Preparing for production

Testing as a development practice



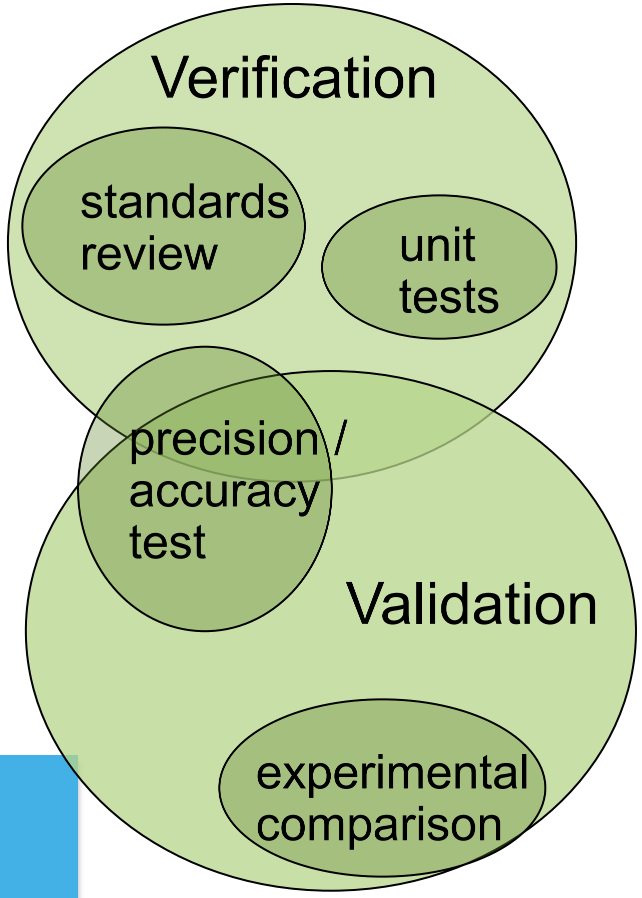
SIAM CSE21, "Querying the ECP" - figshare

Audiences for this presentation

- New to testing / beginning development on a new project
  - Helpful starting points and ways to “start small.”
- Working with a legacy project that needs testing
  - Code isolation for incrementally adding testing
- Improving testing practices on an existing project
  - Ideas and guidelines for a holistic verification strategy

Definitions: Verification vs. Testing vs. Validation

- Software verification addresses design:
  - Does the operational standard make logical sense?
  - Is the implementation consistent with model?
- Model validation checks operation:
  - Is the code capable of handling your target science cases?
  - Is its answer consistent with use expectations?

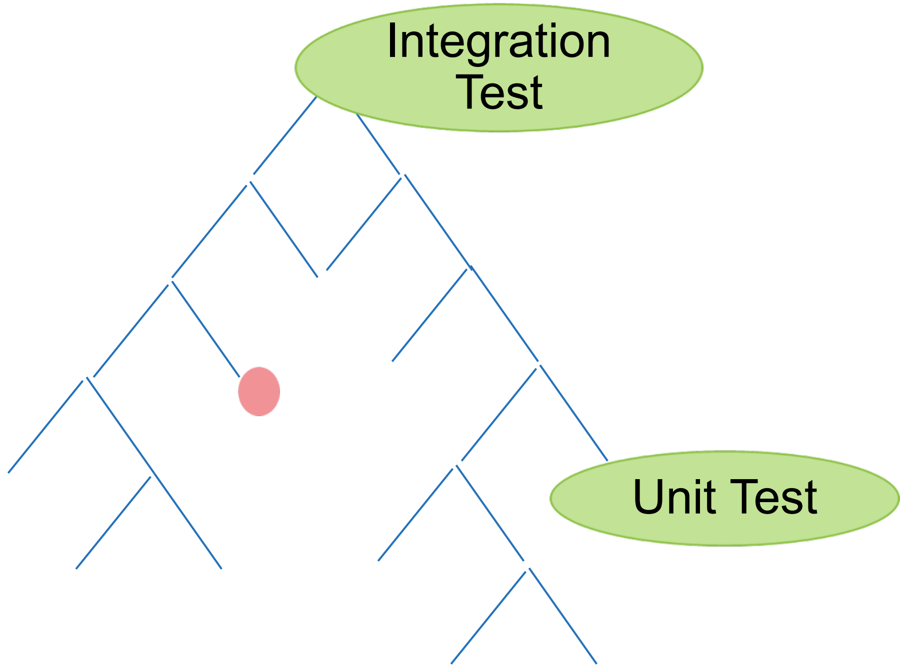


How do verification and validation differ?

- Verification confirms that you have implemented what you meant to
  - Your method does what you wanted it to do
- Validation says whether your science goals are met by your implementation
  - What you wanted your method to do is scientifically valid
  - Your model correctly captures the phenomenon you are trying to understand (outward-looking, not fully captured by tests)

https://www.energy.gov/nnsa/articles/face-uncertainties-nnsa-seeks-verification-and-validation

Components of Verification



- Testing at various granularity levels
  - Individual components
  - Interoperability of components
  - Convergence, stability and accuracy
  - Includes testing "upstream dependencies"
- Validation of individual components
  - Building diagnostics (e.g. ensure conservation of physical quantities)
- Testing practices
  - Error bars
    - Necessary for differentiating between drift and round-off
- Ensuring code and interoperability coverage

Challenges

- Exploratory Software
  - Implies one does not know the outcome
  - Still determining where model is valid
  - A: 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.

Toy Example

```
pip3 install pyscaffold
pip3 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

EXASCALE COMPUTING PROJECT

Toy 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
make –j && make test
```

llnl-blt.readthedocs.io

```
...
[100%] Linking CXX executable ../../../tests/blt_gtest_smoke
[100%] Built target blt_gtest_smoke
mac0103234:build 99r$ 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
```

Going Further

- C, C++, Fortran
  - Running and Reporting Tests: ctest / cdash
  - 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
- 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

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 release 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.