



# Software Testing: Introduction



Greg Watson (he/him)  
Oak Ridge National Laboratory



Developing a Testing and Continuous Integration Strategy for your  
Team tutorial @ Exascale Computing Project Annual Meeting

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



See slide 2 for  
license details

# License, Citation and Acknowledgements

## License and Citation

- This work is licensed under a [Creative Commons Attribution 4.0 International License](#) (CC BY 4.0).
- **The requested citation the overall tutorial is: Gregory R. Watson and David M. Rogers, Developing a Testing and Continuous Integration Strategy for your Team tutorial, in Exascale Computing Project Annual Meeting, online, 2022. DOI: [10.6084/m9.figshare.19608927](#)**
- 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

- What is testing?
- Challenges of testing
- Simple Examples

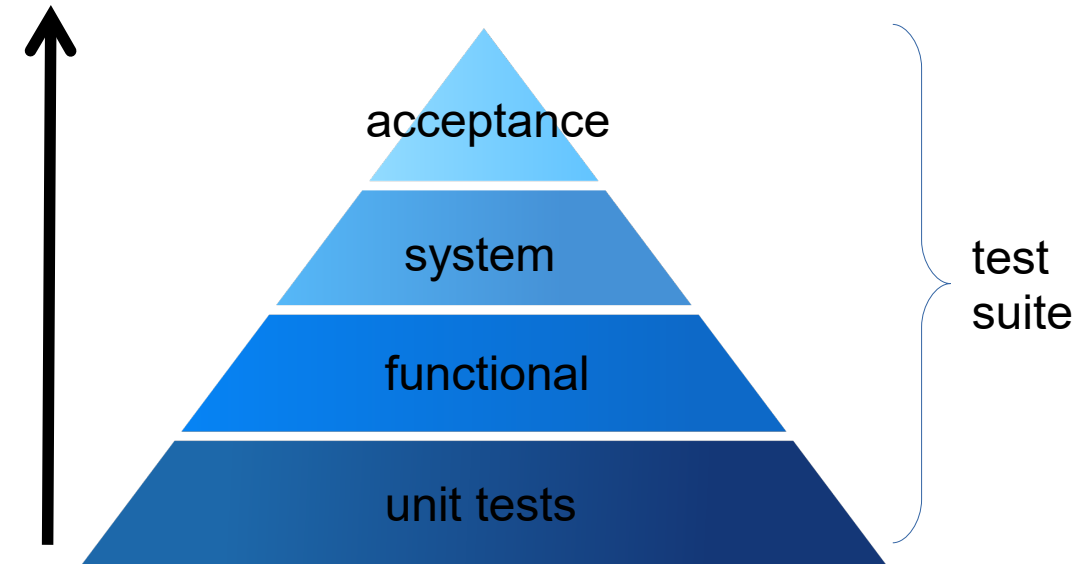
## Testing Examples

- Walk Through Testing Example

# General Categories of Testing

- Development tests
  - Tests run to protect stability while making changes to the code
  - Can include: unit, functional, integration, system, regression, verification, performance, etc.
- Post-installation “smoke” tests
  - Simple tests to ensure the build/install process has succeeded
  - Typically take only a few minutes
  - Could be a *subset* of development tests
- Continuous integration tests
  - Rapid feedback aimed at preventing changes from breaking key branches of the code
  - Run quickly, fail fast, catch problems that would impact other developers
  - Usually associated with automation

Code coverage,  
Complexity



# 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.*

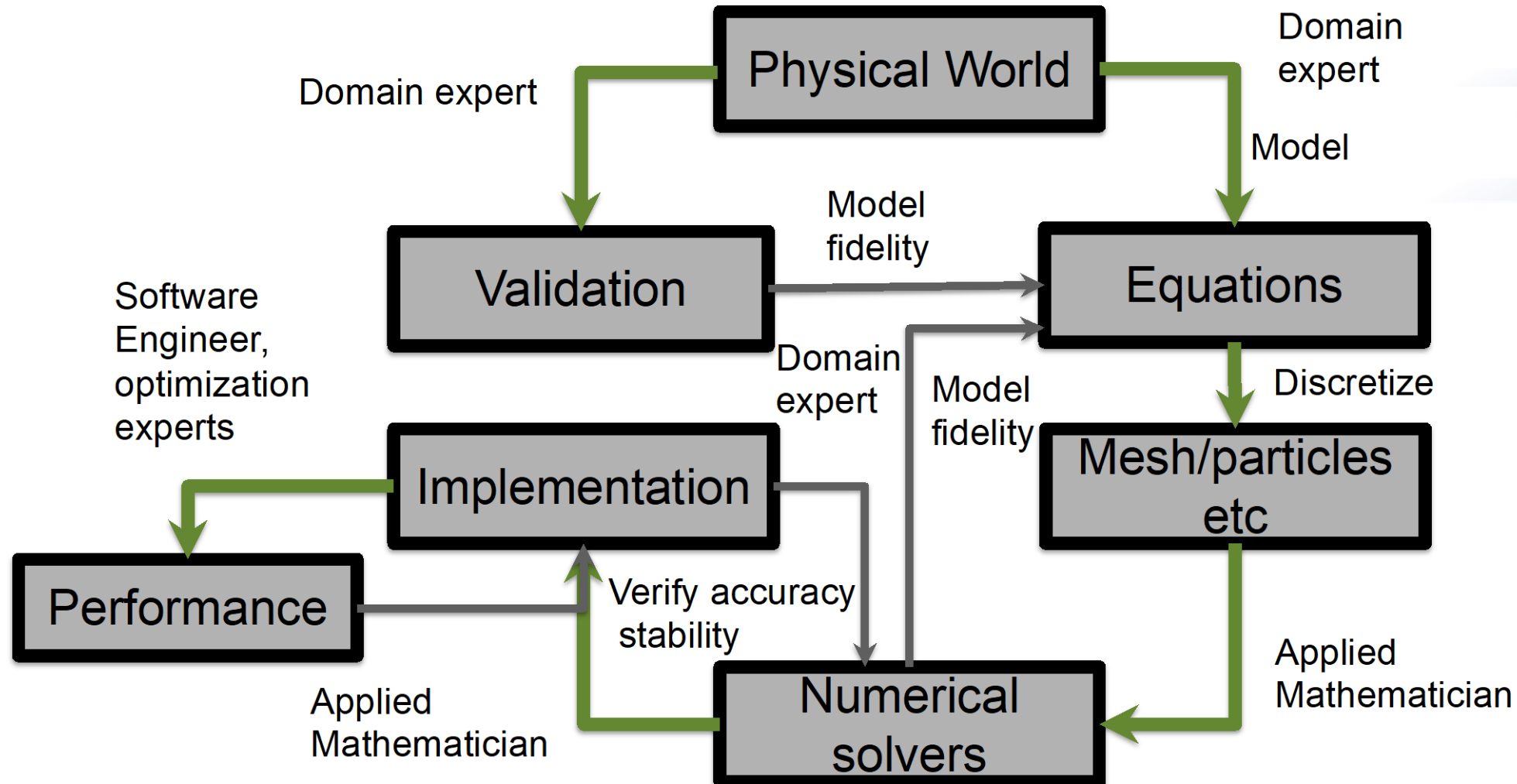
# What about Verification and Validation?

- 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<sup>1</sup> 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

<sup>1</sup>In numerical methods there are times when a single test may not suffice

# Challenges of Testing Complex Software Systems

- Designing tests
  - Complex software tends to have an extensive network of interdependencies
  - For complex scientific software it may be hard to construct a priori tests for some cases
  - Exploratory software implies the outcome is not known or when the model is valid
  - Validation from domain experts feeds back into the design
- Implementing tests
  - Introducing testing into legacy code (legacy == untested)
  - Original verification has been lost in the mists of time.
  - Assumptions, conditions, interactions unknown: “Bad code or necessary evil?”
  - Understanding and progressively improving code coverage

# More Challenges of Testing Complex Software Systems

- Automating tests
  - Just get started – easy to get lost in all the options
  - You must have tests to be able to automate them!
  - Automation does not just mean running every test you have
- What to run where, and when?
  - Consider what resources are required, and what the tests are used for

# How do we determine what 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

# Summary

- A testing strategy is essential for producing reliable trustworthy software
  - Invest the time needed to thoroughly test your software at all levels
  - Use automation whenever possible
- 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.

# 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

# Hands On Activities

- Simple Python and C++ examples
- Checking code coverage example
- Test driven development example

# Python Example

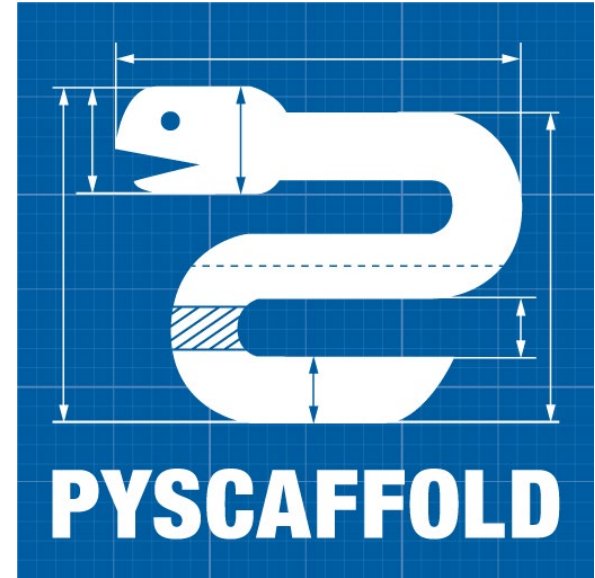
```
$ pip install pyscaffold
$ pip install tox
$ putup autoQCT
$ cd autoQCT # tests in tests/ subdir.
$ tox
```

```
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.org](https://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](https://llnl-blt.readthedocs.io)

# Checking Code Coverage Example

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

- 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	▲	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

265	! Error distribution same <b>for</b> all x values
266	delta = S*Sxx - Sx*Sx
267	<b>if</b> (delta == 0.0_wp) then
268	ERRORMSG("Cannot do linear least-sqrs. Divide by zero.")
269	stop
270	end <b>if</b>
271	delta_inv = 1.0_wp / delta

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

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

# Test Driven Development Example

<https://github.com/bssw-tutorial/hello-numerical-world/tree/main/tdd-example>

```
$ 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
```

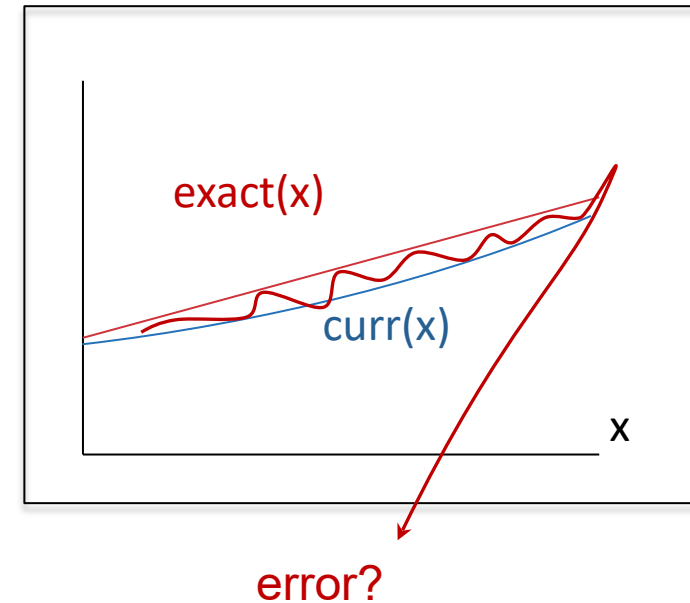
- Lots of setup code – prepares problem for kernel calls
- Isolated, swappable kernel calls
  - Imagine adding kernels to larger, multi-physics application.
- How do we add new kernels using test driven development?

# Build and run the code

```
$ cd tdd-example
$ 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
```

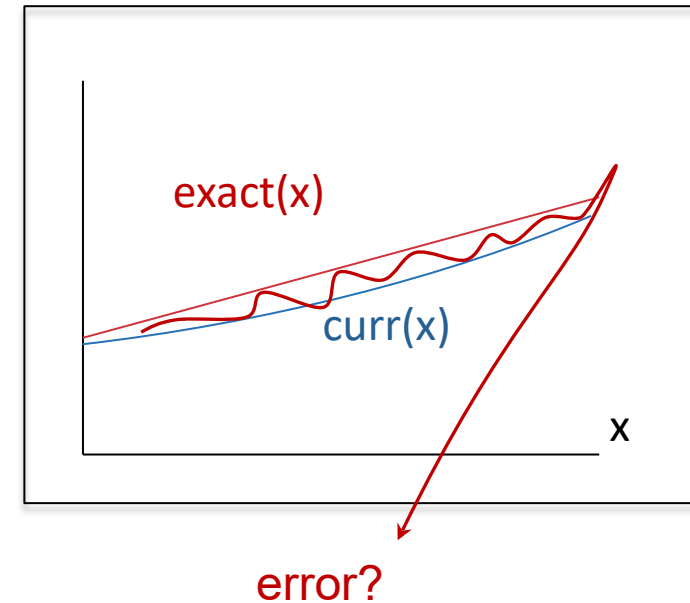
# Add a new kernel

- 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](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

```
$ cd ..
```

- Edit `heat.C`
- Add prototype

```
68
69 extern bool
70 update_solution_upwind15(int n,
71   Double *curr, Double const *last,
72   Double alpha, Double dx, Double dt,
73   Double bc_0, Double bc_1);
```

- Modify assertion

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

- Call kernel

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

# Add new kernel to build

## CMakeLists.txt

```
8 add_executable(heat args.C
9     exact.C
10    heat.C
11    upwind15.C <<< Add new kernel
12    ftcs.C
13    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?

# Resources

- Oberkamp, 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