Evaluating Project Testing Needs

Presented at
Better Scientific Software tutorial

SC17, Denver, Colorado

Anshu Dubey
Argonne National Laboratory

NNSA National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY | Office of Science
IDEAS productivity

License, citation and acknowledgements

License and Citation

- This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).
- Requested citation: Anshu Dubey, Evaluating Project Testing Needs, tutorial, in SC '17: International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colorado, 2017. DOI: 10.6084/m9.figshare.5593348.

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.

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

How to evaluate project needs

And devise a testing regime

Why not always use the most stringent testing?

- Effort spent in devising tests and testing regime are a tax on team resources
- When the tax is too high…
  - Team cannot meet code-use objectives
- When is the tax is too low…
  - Necessary oversight not provided
  - Defects in code sneak through

Evaluating project needs

5

- Objectives: expected use of the code
- Team: size and degree of heterogeneity
- Lifecycle stage: new or production or refactoring
- Lifetime: one off or ongoing production
- Complexity: modules and their interactions

Commonalities

- Unit testing is always good
  - It is never sufficient
- Verification of expected behavior
- Understanding the range of validity and applicability is always important
  - Especially for individual solvers

Development phase – adding on

- Few more steps when adding new components to existing code
  - Know the existing components it interacts with
  - Verify its interoperability with those components
  - Verify that it does not inadvertently break some unconnected part of the code
- May need addition of tests not just for the new component but also for some of the old components
  - This part is often overlooked to the detriment of the overall verification

Selection of tests

- Important to aim for quick diagnosis of error
  - A mix of different granularities works well
    - Unit tests for isolating component or sub-component level faults
    - Integration tests with simple to complex configuration and system level
    - Restart tests
- Rules of thumb
  - Simple
  - Enable quick pin-pointing

Full paper Dubey et al 2015

Approach

- Build a matrix
  - Physics along rows
  - Infrastructure along columns
  - Alternative implementations, dimensions, geometry
- Mark <i,j> if test covers corresponding features
- Follow the order
  - All unit tests – including full module tests
  - Tests representing ongoing productions
  - Tests sensitive to perturbations
  - Most stringent tests for solvers
  - Least complex test to cover remaining spots

Example

	Hydro	EOS	Gravity	Burn	Particles
AMR	CL	CL		CL	CL
UG	SV	SV			SV
Multigrid	WD	WD	WD	WD	
FFT			PT		

Tests	Symbol
Sedov	SV
Cellular	CL
Poisson	PT
White Dwarf	WD

- A test on the same row indicates interoperability between corresponding physics
- Similar logic would apply to tests on the same column for infrastructure
- More goes on, but this is the primary methodology

Refactoring

Testing needs during code refactor

Considerations

- Know why you are refactoring
- Know the scope of refactoring
- Know bounds on acceptable behavior change
- Know your error bounds
  - Bitwise reproduction of results unlikely after transition
- Map from here to there
- Check for coverage provided by existing tests
- Develop new tests where there are gaps

Incorporate testing overheads into refactor cost estimates

Challenges with legacy codes

Checking for coverage

- Legacy codes can have many gotchas
  - Dead code
  - Redundant branches
- Interactions between sections of the code may be unknown
- Can be difficult to differentiate between just bad code, or bad code for a good reason
  - Nested conditionals

Code coverage tools are of limited help

An Approach

- Isolate a small area of the code
- Dump a useful state snapshot
- Build a test driver
  - Start with only the files in the area
  - Link in dependencies
    - Copy if any customizations needed
- Read in the state snapshot
- Verify correctness
  - Always inject errors to verify that the test is working

Methodology developed for the ACME project, proving to be very useful

Agenda

Tutorial evaluation form: http://bit.ly/sc17-eval

Time	Topic	Speaker
8:30am-8:45am	Why effective software practices are essential for CSE projects	David E. Bernholdt, ORNL
8:45am-9:15am	Introduction to software licensing	David E. Bernholdt, ORNL
9:15am-9:45am	Better (small) scientific software teams	Michael A. Heroux, SNL
9:45am-10:00am	Improving Reproducibility Through Better Software Practices	Michael A. Heroux, SNL
10:00am-10:30am	*Break*	
10:30am-10:45am	Testing of HPC Scientific Software: Introduction	Alicia M. Klinvex, SNL
10:45am-11:15am	Verification	Anshu Dubey, ANL
11:15am-11:45am	Evaluating project testing needs	Anshu Dubey, ANL
11:45am-12:00pm	Code coverage demo and CI demo	Alicia M. Klinvex, SNL

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT