SOFTWARE PRACTICES IN COMPUTATIONAL SCIENCE COMMUNITIES – AN OVERVIEW

OCTOBER 13, 2016

ANSHU DUBEY
MATHEMATICS AND COMPUTER SCIENCE DIVISION
ARGONNE NATIONAL LABORATORY

OUTLINE

- ❑ **Motivation**
- ❑ Customization
- ❑ Best Practices

HEROIC PROGRAMMING

Usually a pejorative term, is used to describe the expenditure of huge amounts of (coding) effort by talented people to overcome shortcomings in process, project management, scheduling, architecture or any other shortfalls in the execution of a software development project in order to complete it. Heroic Programming is often the only course of action left when poor planning, insufficient funds, and impractical schedules leave a project stranded and unlikely to complete successfully.

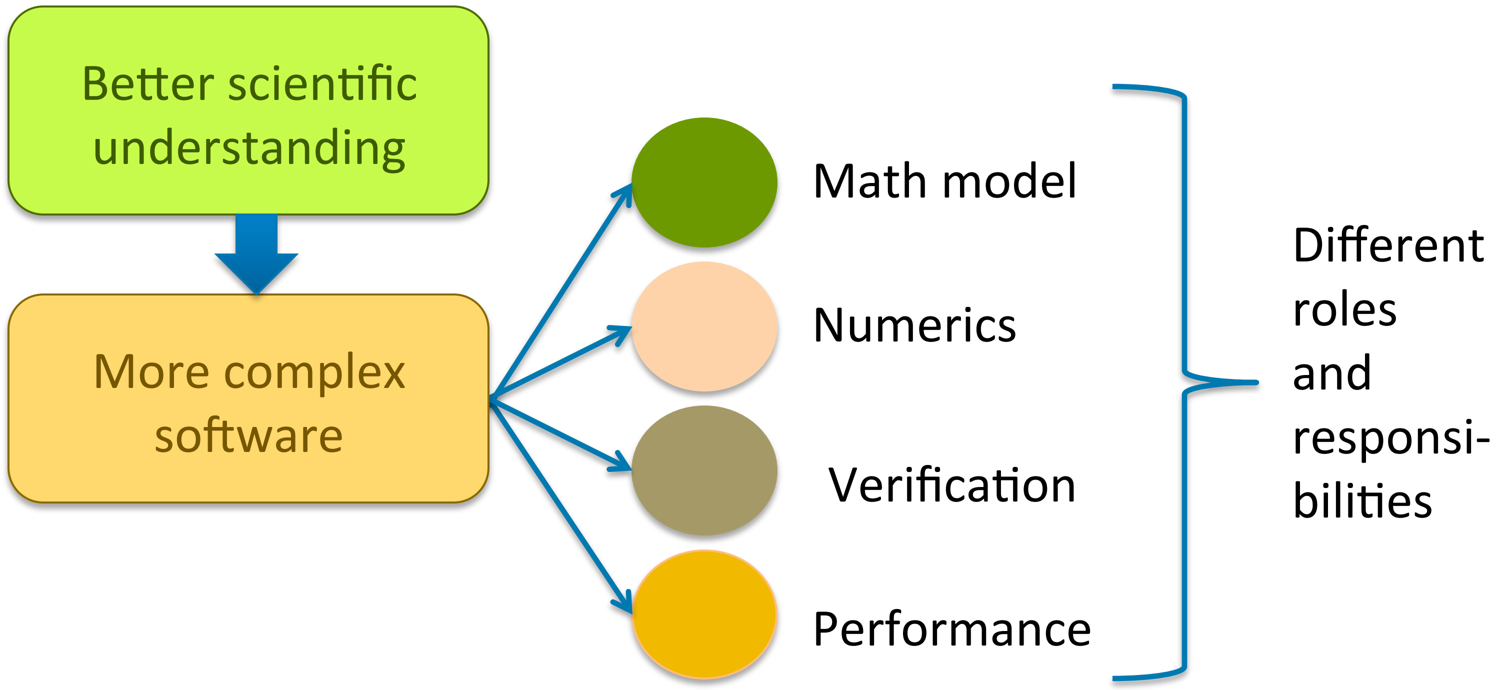
From http://c2.com/cgi/wiki?HeroicProgramming

Science teams often resemble heroic programming

Many do not see anything wrong with that approach

WHAT IS WRONG WITH HEROIC PROGRAMMING

Scientific results that could be obtained with heroic programming have run their course, because:


Better scientific understanding
More complex software
Math model
Numerics
Verification
Performance
Different roles and responsi-bilities

It is not possible for a single person to take on all these roles

GENERAL STATE OF SCIENTIFIC CODES

- ❑ Start in small groups
- ❑ Accretion leads to unmanageable software
- ❑ Parts of software may become unusable over time
- ❑ Inadequately verified software produces questionable results
- ❑ Increases ramp-on time for new developers
- ❑ Reduces software and science productivity due to technical debt

Technical debt – implementation without design and plan (quick and dirty) collects interest => more effort required to add features later.

Debt can compound

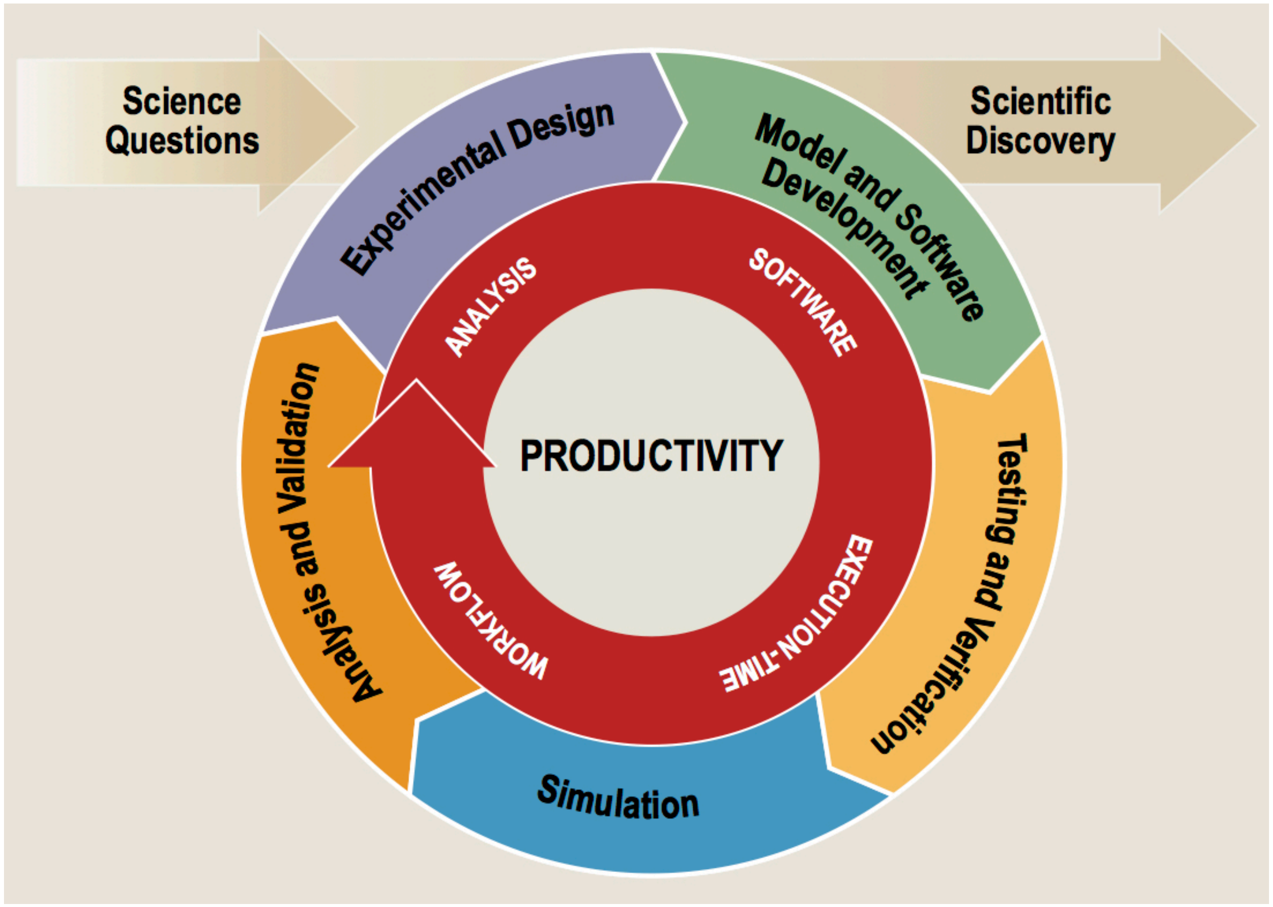
LOOKING TOWARD FUTURE

- ❑ Codes aiming for higher fidelity modeling
  - ❑ More complex codes, simulations and analysis
  - ❑ Numerous models, more moving parts that need to interoperate
  - ❑ Variety of expertise needed – the only tractable development model is through separation of concerns
  - ❑ **It is more difficult to work on the same software in different roles without a software engineering process**
- ❑ Onset of higher platform heterogeneity
  - ❑ Requirements are unfolding, not known apriori
  - ❑ **The only safeguard is investing in flexible design and robust software engineering process**

"... it seems likely that significant software contributions to existing scientific software projects are not likely to be rewarded through the traditional reputation economy of science. Together these factors provide a reason to expect the over-production of independent scientific software packages, and the underproduction of collaborative projects in which later academics build on the work of earlier ones."

Howison & Herbsleb (2011)

SOFTWARE PRODUCTIVITY CYCLE

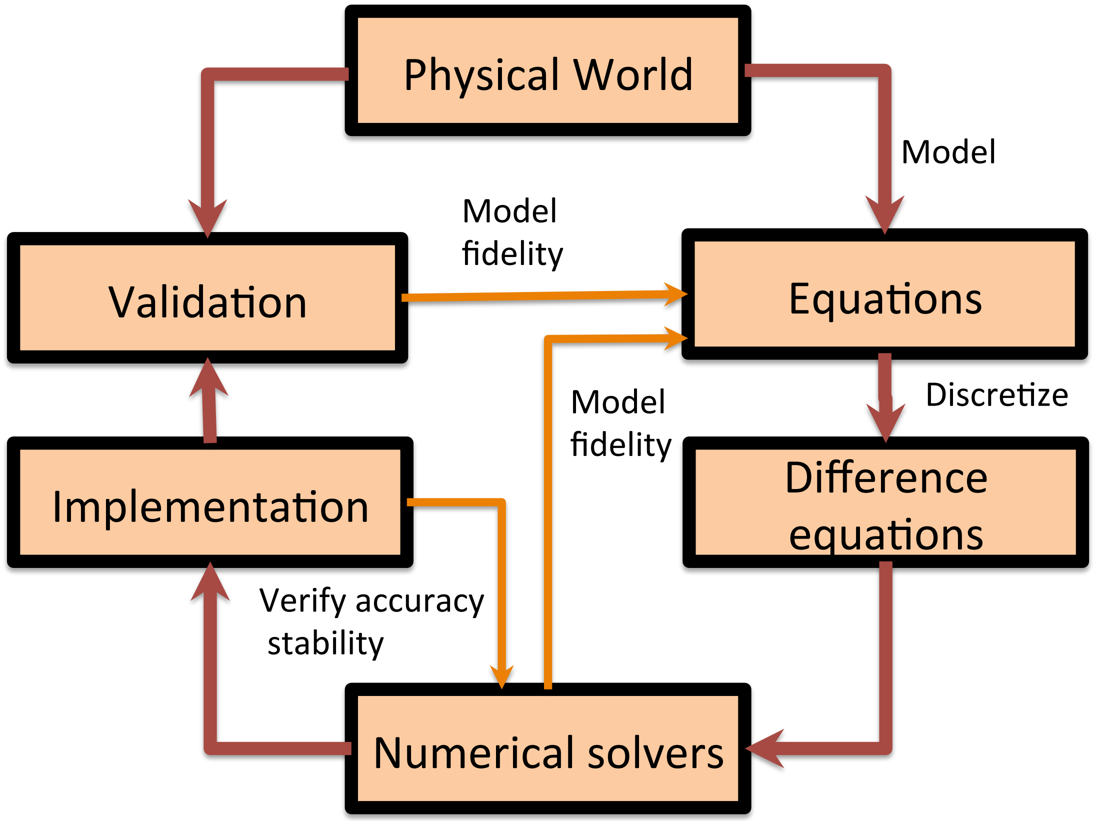


http://www.orau.gov/swproductivity2014/SoftwareProductivityWorkshopReport2014.pdf

OUTLINE

- ❑ Motivation
- ❑ **Customization**
- ❑ Best Practices

LIFECYCLE



- ❑ Modeling
  - ❑ Approximations
  - ❑ Discretizations
  - ❑ Numerics
    - ❑ Convergence
    - ❑ Stability
- ❑ Implementation
  - ❑ Verification
    - ❑ Expected behavior
  - ❑ Validation
    - ❑ Experiment/ observation

GENERAL CHALLENGES

Technical

- ❑ All parts can be under research
- ❑ Knowledge growth => change in requirements
- ❑ Real world is messy, so is the software

Sociological

- ❑ Competing priorities and incentives
- ❑ Limited resources
- ❑ Perception of overhead without benefit
- ❑ Interdisciplinary interactions

VALIDATION CHALLENGES

- Interdisciplinary
  - Domain knowledge
  - Applied mathematics
  - Software engineering
- Exploring uncharted territories
  - Existing knowledge is of limited interest
  - Need to push the boundaries
  - The behavior of solvers not always predictable in regimes of interest

VERIFICATION CHALLENGES

- Inadequate granularity definition
  - Especially in composable codes
- Code coverage gives incomplete picture
  - Interoperability coverage as important
- Legacy components
  - No existing tests of any granularities
  - Examples – multiphysics application codes that support multiple domains

TESTING CHALLENGES

- Testing needs differ
    - Extent and granularity
    - Degree of formalization
    - Floating point issues
        - Different results
            - On different platforms and runs
            - Ill-conditioning can magnify these small differences
                - Final solution may be different
            - Number of iterations may be different
    - Unit testing
        - Isolating behavior can be difficult

CONSIDERATIONS FOR CUSTOMIZATION

- ❑ There is no “all or none”
  - ❑ Focus on improving productivity
  - ❑ Minimize bias

- ❑ Fine balance between buy-in and imposition
  - ❑ Show benefit to convert
    - ❑ Overcome resistance to change
    - ❑ Allay suspicion of new processes

EVALUATE PROJECT NEEDS

- ❑ Objectives
  - ❑ Proof of concept
  - ❑ Limited research use
  - ❑ Library
  - ❑ Production – simulations and analysis
- ❑ Team
  - ❑ Number of developers
  - ❑ Background of developers
  - ❑ Geographical spread

EVALUATE PROJECT NEEDS

- ❑ Lifecycle stages
- ❑ Lifetime
  - ❑ How long a code is expected to live
  - ❑ New code versus some legacy components
- ❑ Complexity
  - ❑ Number of modules, models, data structures, solvers
  - ❑ Degree of coupling and interoperability requirements

OUTLINE

- ❑ Motivation
- ❑ Customization
- ❑ **Best Practices**

BASELINE

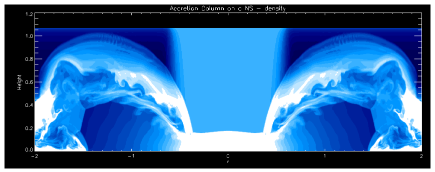
- ❑ Customize process
- ❑ Invest in code design
- ❑ Use version control and automated testing
- ❑ Institute appropriate verification and validation regime
- ❑ Define coding and testing standards
- ❑ Clear and well defined policies for
    - ❑ Auditing and maintenance
    - ❑ Distribution and contribution
    - ❑ Documentation

DESIRABLE

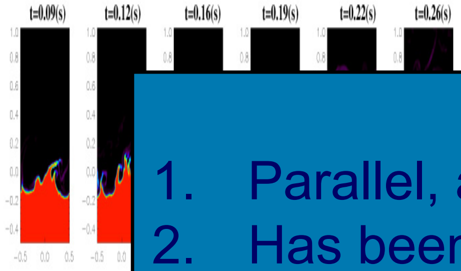
- ❑ Provenance and reproducibility
- ❑ Lifecycle management
- ❑ Open development and frequent releases

EXAMPLE : FLASH

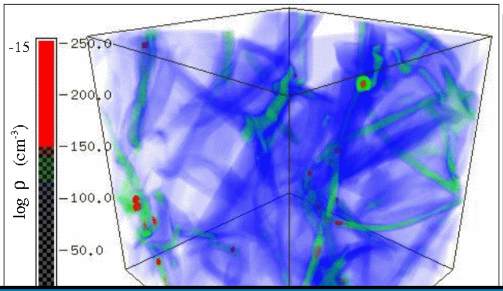
1. Parallel, adaptive-mesh refinement (AMR) code
2. Has been in public release for 16 years
3. Can solve a broad range of problems
4. Fully modular and extensible: components can be combined to create many different applications
5. Serves half a dozen research communities

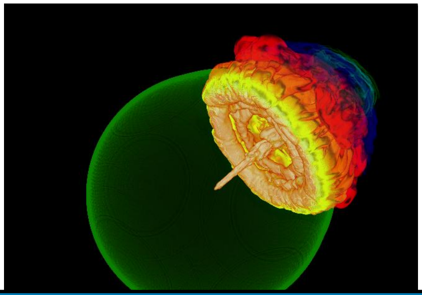


Shortly: Relativistic accretion onto NS

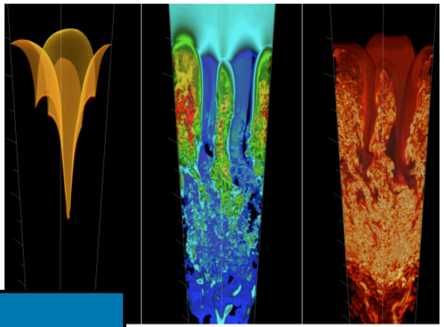


Wave break

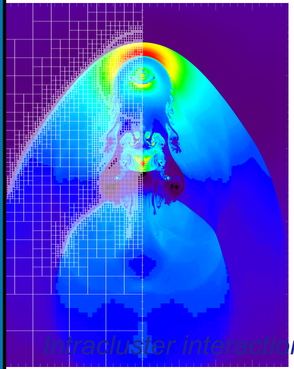




Nova outbursts on white dwarfs

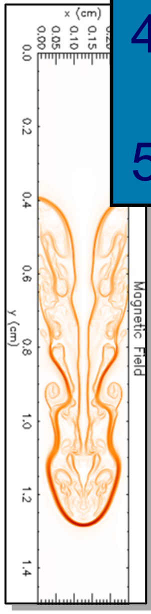


clear Burning

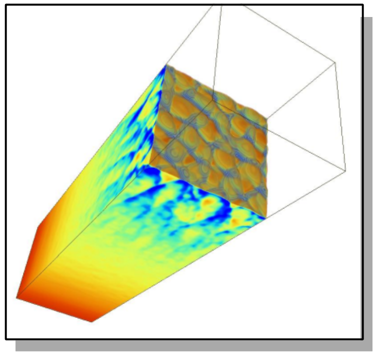


Intracluster interactions

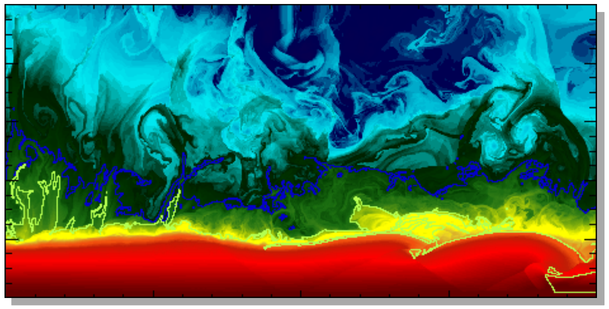
Rayleigh-Taylor instability



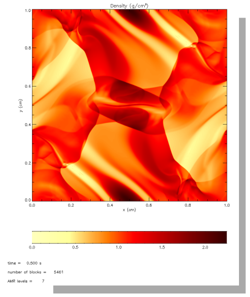
Magnetic Rayleigh-Taylor



Cellular detonation



Helium burning on neutron stars



Orzag/Tang MHD vortex

Richtmyer-Meshkov instability

AUDITING PROCESS

- ❑ SVN for Version Control
  - ❑ Production branches for different projects
- ❑ Online Coding Violation Tracking and Bugzilla
  - ❑ Unfinished tasks, bugs, bad code, developer queries
- ❑ Profiling Tools
  - ❑ Memory / speed diagnostic tools
- ❑ Documentation
  - ❑ Online documentation
  - ❑ User’s guide in HTML and PDF
  - ❑ Online developers guide
- ❑ User Support
  - ❑ Email users’ group
  - ❑ Periodic tutorials (presentations remain online)

INTERDISCIPLINARY INTERACTIONS

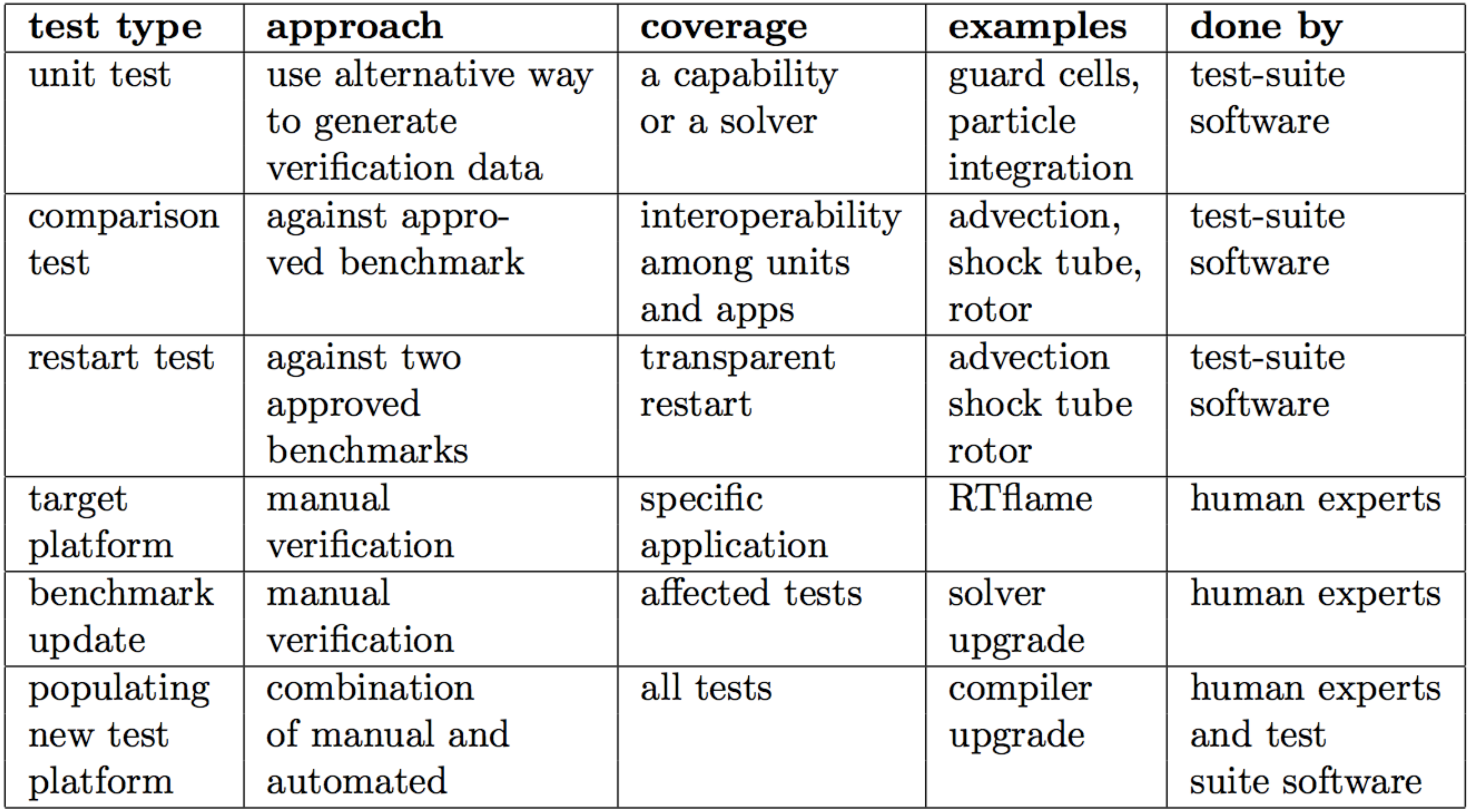
A partnership model that works

- ❑ Science users treat the code as a research instrument that needs its own research
- ❑ Developers and computer scientists interested in a product and the science being done with the code
  - ❑ Helps to have people with multidisciplinary training
- ❑ Comparable resources and autonomy for the developers
  - ❑ And recognition of their intellectual contribution to scientific discovery
- ❑ Careful balance between long term and short term objectives

CONTRIBUTION AND ATTRIBUTION POLICIES

- ❑ Balancing act between IP protection and openness
  - ❑ Pre-negotiated period of time when the code exists in FLASH repo but is not released
- ❑ The contribution includes tests
- ❑ At least one example setup and documentation
- ❑ All contributions are acknowledged in user's guide and release notes
- ❑ The contributors can also provide publications to be cited

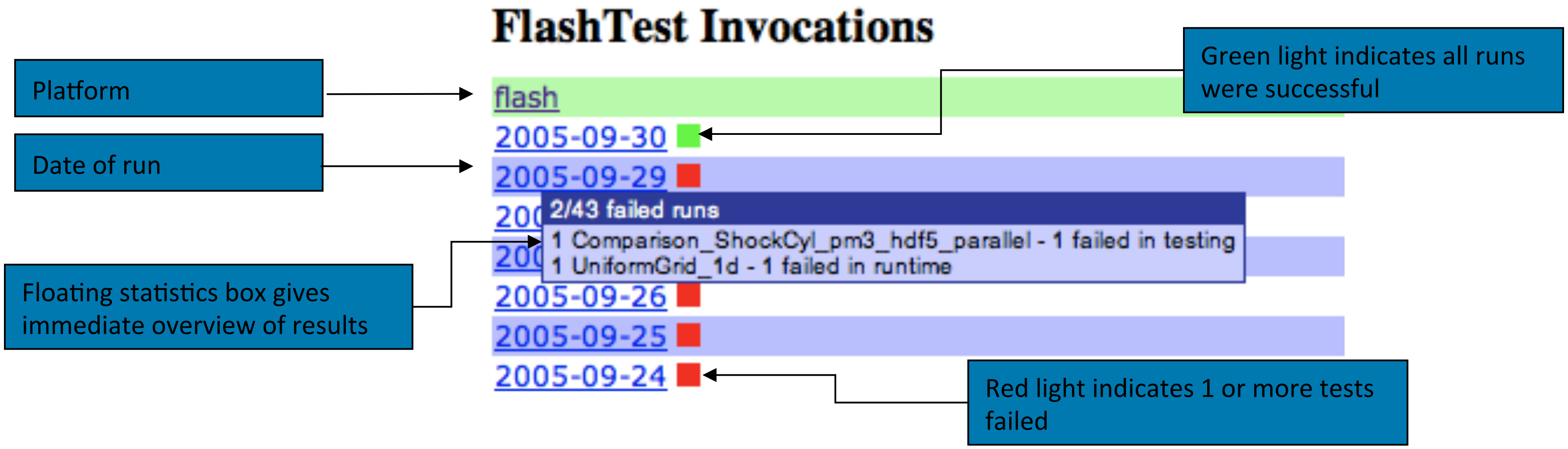
THE TESTS COLLECTION



test type	approach	coverage	examples	done by
unit test	use alternative way to generate verification data	a capability or a solver	guard cells, particle integration	test-suite software
comparison test	against appro-ved benchmark	interoperability among units and apps	advection, shock tube, rotor	test-suite software
restart test	against two approved benchmarks	transparent restart	advection shock tube rotor	test-suite software
target platform	manual verification	specific application	RTflame	human experts
benchmark update	manual verification	affected tests	solver upgrade	human experts
populating new test platform	combination of manual and automated	all tests	compiler upgrade	human experts and test suite software

THE TEST SUITE

- Runs a variety of problems on multiple platforms on a daily basis
- A platform is defined as a combination of hardware, OS and compiler suite
- In-house software manages automated runs
- Also provides web interface for inspection and modification of tests



FlashTest Invocations

flash

2005-09-30 🟩

2005-09-29 🟥

200

200

2005-09-26 🟥

2005-09-25 🟥

2005-09-24 🟥

2/43 failed runs

1 Comparison_ShockCyl_pm3_hdf5_parallel - 1 failed in testing

1 UniformGrid_1d - 1 failed in runtime

Platform
Date of run
Floating statistics box gives immediate overview of results
Green light indicates all runs were successful
Red light indicates 1 or more tests failed

FlashTest

CONCLUSIONS

- There are many reasons why software engineering practices are good and should be encouraged
  - Science and engineering by simulation needs more scrutiny into the methods and software
  - There is no need to keep reinventing the wheel
    - This is especially true of book-keeping work
    - Reuse infrastructural components
- The days of heroic programming are past, collaborative efforts are more productive
- **They are indispensible for large-scale computing**

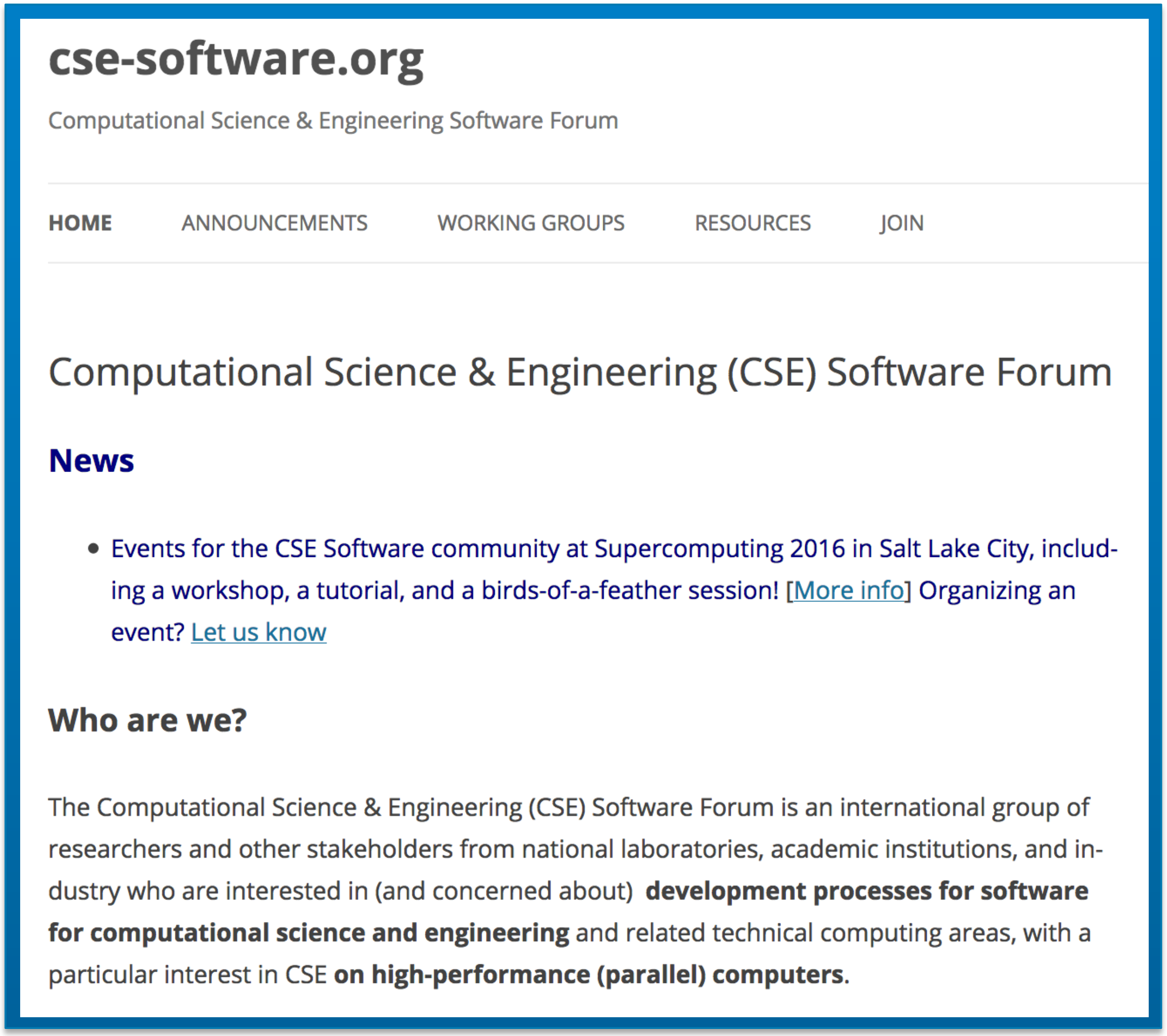
It is extremely important to recognize that science through computing is only as good as the software that produces it

USEFUL RESOURCES

https://ideas-productivity.org/resources/howtos/

- **'What Is' docs**: 2-page characterizations of important topics for SW projects in computational science & engineering (CSE)
- **'How To' docs**: brief sketch of best practices
  - Emphasis on ``bite-sized" topics enables CSE software teams to consider improvements at a small but impactful scale
- We welcome feedback from the community to help make these documents more useful

cse-software.org: Join the community


cse-software.org

Computational Science & Engineering Software Forum

Computational Science & Engineering (CSE) Software Forum

News

- Events for the CSE Software community at Supercomputing 2016 in Salt Lake City, including a workshop, a tutorial, and a birds-of-a-feather session! [More info] Organizing an event? Let us know

Who are we?

The Computational Science & Engineering (CSE) Software Forum is an international group of researchers and other stakeholders from national laboratories, academic institutions, and industry who are interested in (and concerned about) **development processes for software for computational science and engineering** and related technical computing areas, with a particular interest in CSE **on high-performance (parallel) computers**.

Contribute to and share activities and resources for CSE software productivity and software engineering

OTHER RESOURCES

http://www.software.ac.uk/

http://software-carpentry.org/

http://flash.uchicago.edu/cc2012/

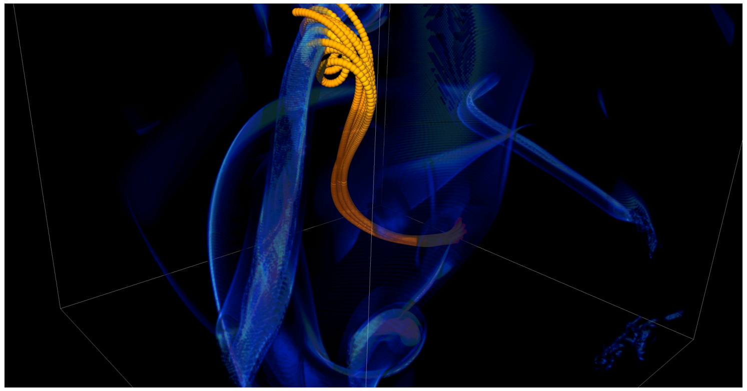
http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745

http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=4375255

http://www.orau.gov/swproductivity2014/
SoftwareProductivityWorkshopReport2014.pdf

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6171147

LAST THOUGHT: WHAT CAN HAPPEN WHEN PROCESS IS IGNORED



- In 2005 BG/L was made available at short notice
- Quick and dirty development of particles
- Many in-flight corrections of defects
- One was adding tags to track individual particles
  - **Got many duplicated tags due to round-off**
- Had to develop post-processing tools to correctly identify trajectories

FLASH had a software process
Process was short circuited due to time constraints

We got ready for the run in less than a month, the run went for 1.5 weeks, and it took over 6 months before we could trust the results.

SURVEY OF IDEAS USE-CASES

IDEAS scientific software productivity project:
www.ideas-productivity.org

- ❑ Five application codes and four numerical libraries
- ❑ All use version control, and all but one use distributed version control
- ❑ Builds are evenly divided between GNU make and CMake
- ❑ All provide documentation with some form of user’s guide, many use automated documentation generation tools
- ❑ All have testing in some form, a couple do manual regression testing, the rest are automated
- ❑ Roughly half make use of unit testing explicitly
- ❑ Majority are publicly available

SUMMARY FROM COMMUNITY CODES WORKSHOP (2012)

http://flash.uchicago.edu/cc2012/

- Codes – FLASH, Cactus, Enzo, ESMF, Lattice QCD code-suite, AMBER, Chombo, and yt
- Software architecture is almost always in the form of composable components
  - Need for extensibility
- All codes have rigorous auditing processes in place
- Gatekeeping for contributions, though models are different
- All codes have wide user communities, and the communities benefit from a common highly exercised code base