better scientific software

Improving Reproducibility Through Better Software Practices

Presented by

PESO https://pesoproject.org

RAPIDS https://rapids.lbl.gov

Members of

CASS

Consortium for the Advancement of Scientific Software
https://cass.community

With prior support from

ECP EXASCALE COMPUTING PROJECT

IDEAS productivity

David E. Bernholdt (he/him)
Oak Ridge National Laboratory

Software Sustainability track @ Argonne Training Program on Extreme-Scale Computing summer school

Contributors: David E. Bernholdt (ORNL), Anshu Dubey (ANL), Patricia A. Grubel (LANL), Michael A. Heroux (SNL), Jared O'Neal (ANL), David M. Rogers (ORNL), Gregory R. Watson (ORNL)

Additional thanks to: Juan Pablo Haddad, Akash Dhruv, Steve Fickas, Carlo Graziani, Boyana Norris

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is:** Anshu Dubey, David E. Bernholdt, and Todd Gamblin, Software Sustainability track, in Argonne Training Program on Extreme-Scale Computing, St. Charles, Illinois, 2025. DOI: 10.6084/m9.figshare.29816981.
- Individual modules may be cited as *Speaker, Module Title*, in *Tutorial Title*, …

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 supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies (NGSST) program.
- 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 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 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 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.

Terminology

A few of the terms used when talking about this topic…

Reproducibility

Replicability

Reliability

Correctness

Accuracy

Transparency

Credibility

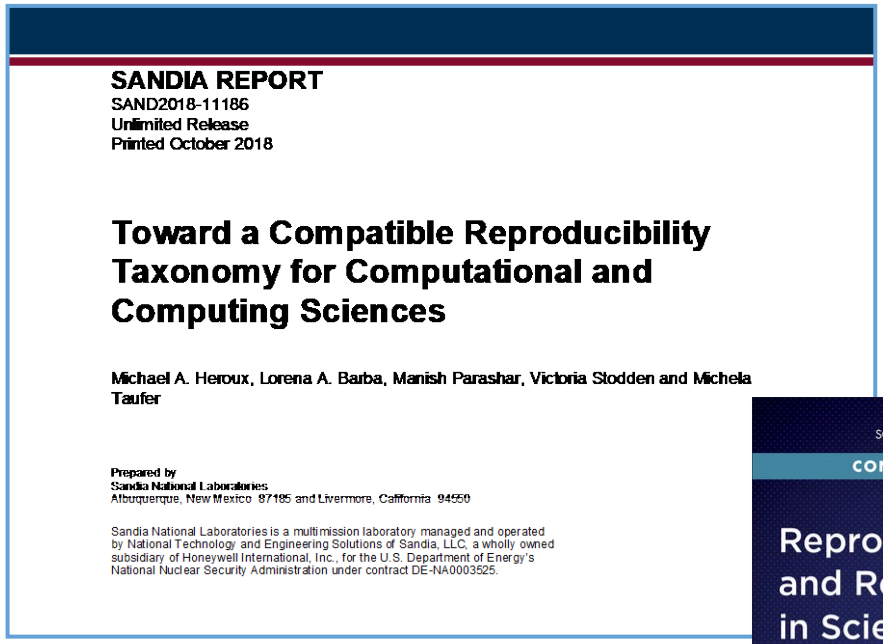
They don’t mean exactly the same thing…

…but for the purposes of this presentation, the differences don’t really matter

Reproducible vs Replicable: A Special Note

- Historically different communities have defined these two differently
- With the increased focus, there has also been an effort to unify the language
- Consensus around the definitions of Claerbout, et al.
  - Others are in the process of switching their terminology to match, i.e., ACM

- **Reproducible**: Another team is able to obtain the same result using the authors' experimental environment
- **Replicable**: Another team is able to obtain consistent results using a different experimental environment



SANDIA REPORT
SAND2018-11186
Unlimited Release
Printed October 2018

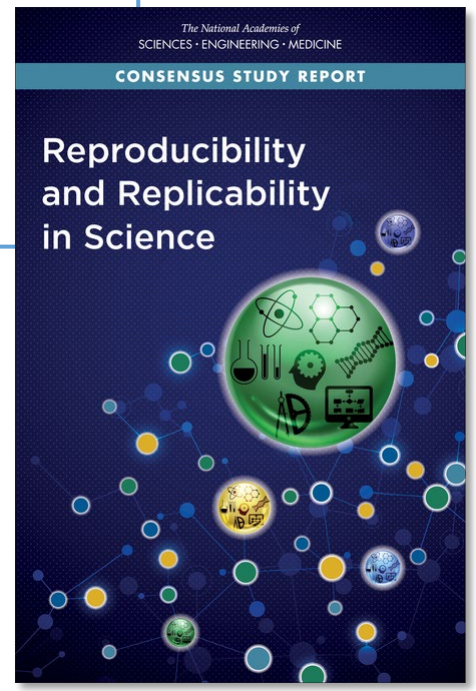
Toward a Compatible Reproducibility Taxonomy for Computational and Computing Sciences

Michael A. Heroux, Lorena A. Barba, Manish Parashar, Victoria Stodden and Michela Taufer

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission 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.

doi:10.2172/1481626



doi:10.17226/25303

Why Reproducibility is Important

Transparency & Reproducibility

SCIENCE

Many Psychology Findings Not as Strong as Claimed

By BENEDICT CAREY AUG. 27, 2015

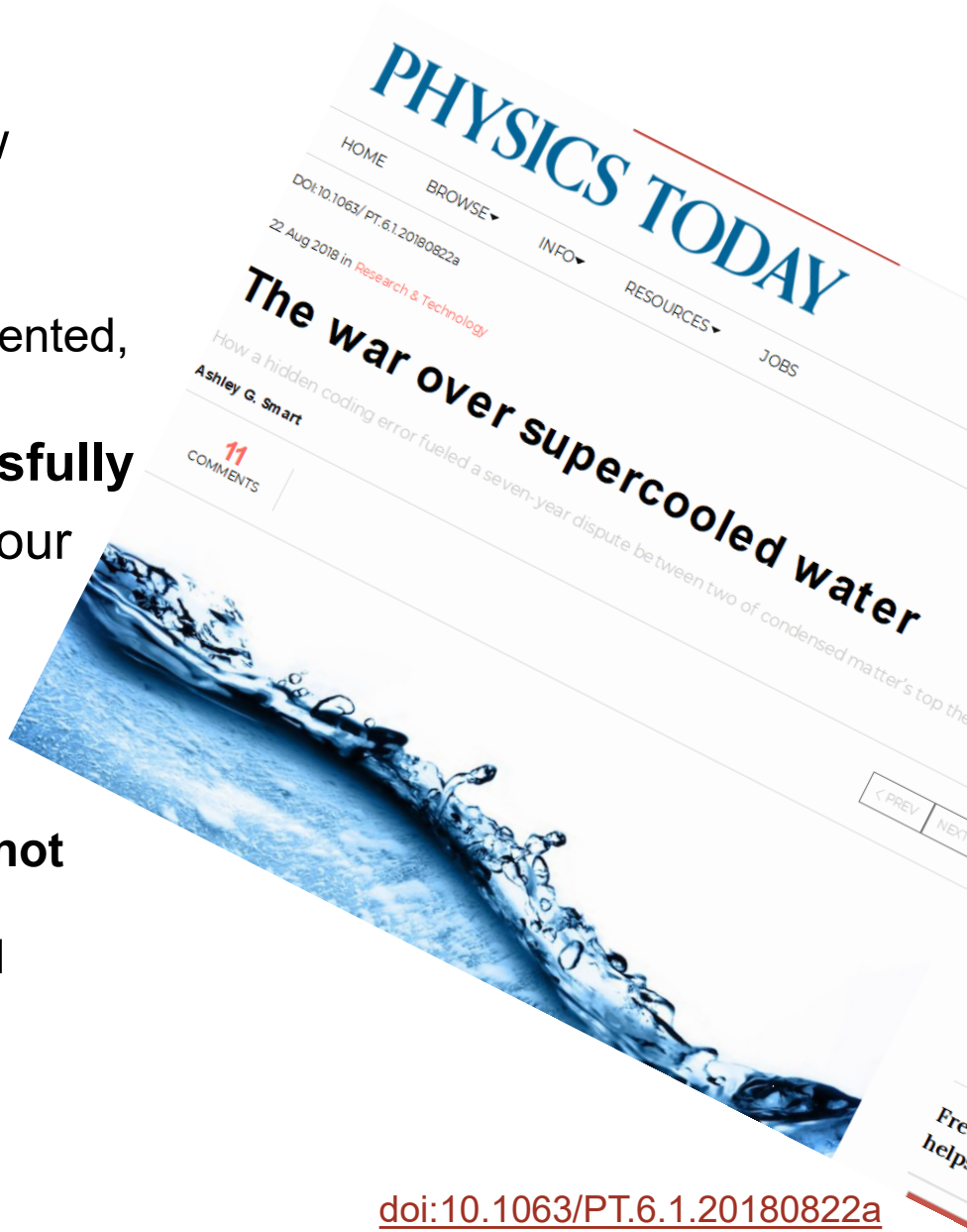
Staff of the the Reproducibility Project at the Center for Open Science in Charlottesville, Va., from left: Mallory Kidwell, Courtney Soderberg, Johanna Cohoon and Brian Nosek. Dr. Nosek and his team led an attempt to replicate the findings of 100 social science studies. Andrew Shurtleff for The New York Times

- NY Times highlights “problems”.
- Only one of many cited examples.
- Computational science ***had*** been spared this “spotlight”.

http://www.nytimes.com/2015/08/28/science/many-social-science-findings-not-as-strong-as-claimed-study-says.html

Example: Behavior of pure water just above homogeneous nucleation temperature (~ - 40 °C/°F)

- Debenedetti/Princeton (2009): 2 possible phases: High or low density
- Chandler/Berkeley (2011): Only 1 phase: High density
  - "LAMMPS codes used in refs 5 and 12 are standard and documented, with scripts freely available upon request."
- Debenedetti tries to reproduce Chandler's results, **unsuccessfully**
- Debenedetti (with colleague Palmer) to Chandler: "Send us your code"
  - Chandler responded only after intervention by editor of Nature
- Palmer identified a bug/feature in Chandler's code: a change intended merely to speed up the code gave **different results**
  - Code was not actually "standard and documented"; **paper was not reproducible**
  - When replaced with a more standard approach, results matched Debenedetti's
- Took **seven years** to resolve!
- *Would testing have caught this?*



doi:10.1063/PT.6.1.20180822a

Example: Python Behaves Differently on Different Platforms

- Scripts for analyzing experimental nuclear magnetic resonance (NMR) data
- Scripts use Python's glob module (listing filenames matching a pattern)
- Module ordered results differently in Linux and Mac Mojave
- Results depended on the order in which files were processed
- **Casts doubt on results in 150 papers**
- *Would testing have caught this?*

https://arstechnica.com/information-technology/2019/10/chemists-discover-cross-platform-python-scripts-not-so-cross-platform/

Science through computing is,
at best,
as credible as the software that produces it!

Incentives for Paying Attention to Reproducibility

Common statement: “I would love to do a better job on my software, but I need to:

- Get this paper submitted
- Complete this project task
- Do something my employer values more

Goal: Change incentives to include valuing of better software, better science

This is a long-term goal, requiring a culture change in (computational) science which is in the early stages

Funders and the Community Setting Expectations for Your Data (Including Your Software)

Data Management Plans

- Most research sponsors require data management plans as part of proposals
- "Data" includes software (increasingly called out explicitly)
- Example: NSF policy on Dissemination and Sharing of Research Results
  - Investigators can keep their legal rights over their intellectual property, but they still have to make their results, data, and collections available to others
  - Policies will be implemented via
    - Proposal review
    - Award negotiations and conditions
    - Support/incentives

FAIR Principles for Research Software

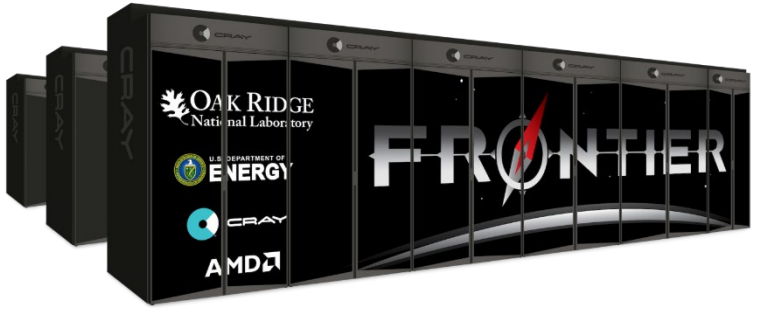
- How to make all research outcomes, including software, optimally reusable by humans and machines?
- Findability
  - Software is described with rich metadata and a unique and assigned a persistent identifier
- Accessibility
  - Metadata are retrievable by their identifier, even when the software is no longer available
- Interoperability
  - Software exchanges data in a way that meets domain-relevant community standards
- Reusability
  - Software has a clear license, detailed provenance, and meets domain-relevant community standards
- Also FAIR Principles for Data

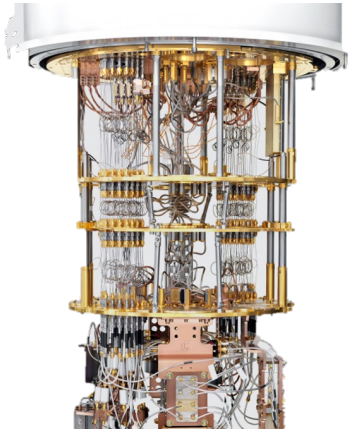
Increasing Attention on Reproducibility from Publishers

- More publication venues are adding reproducibility **recognition** or **requirements**
- ACM ~~Replicated~~ Reproducible Computational Results (RCR)
  - ACM TOMS, TOMACS
  - http://toms.acm.org/replicated-computational-results.cfm
- ACM Badging
  - Functional, reusable, available, replicated, reproduced
  - https://www.acm.org/publications/policies/artifact-review-and-badging-current
- These conferences have artifact evaluation appendices:
  - CGO, PPoPP, PACT, RTSS and SC
  - http://fursin.net/reproducibility.html
- NISO Committee on Reproducibility and Badging
  - https://www.niso.org/niso-io/2019/01/new-niso-project-badging-scheme-reproducibility-computational-and-computing
  - Publishers: ACM, IEEE, figshare, STM, Reed Elsevier, Springer Nature

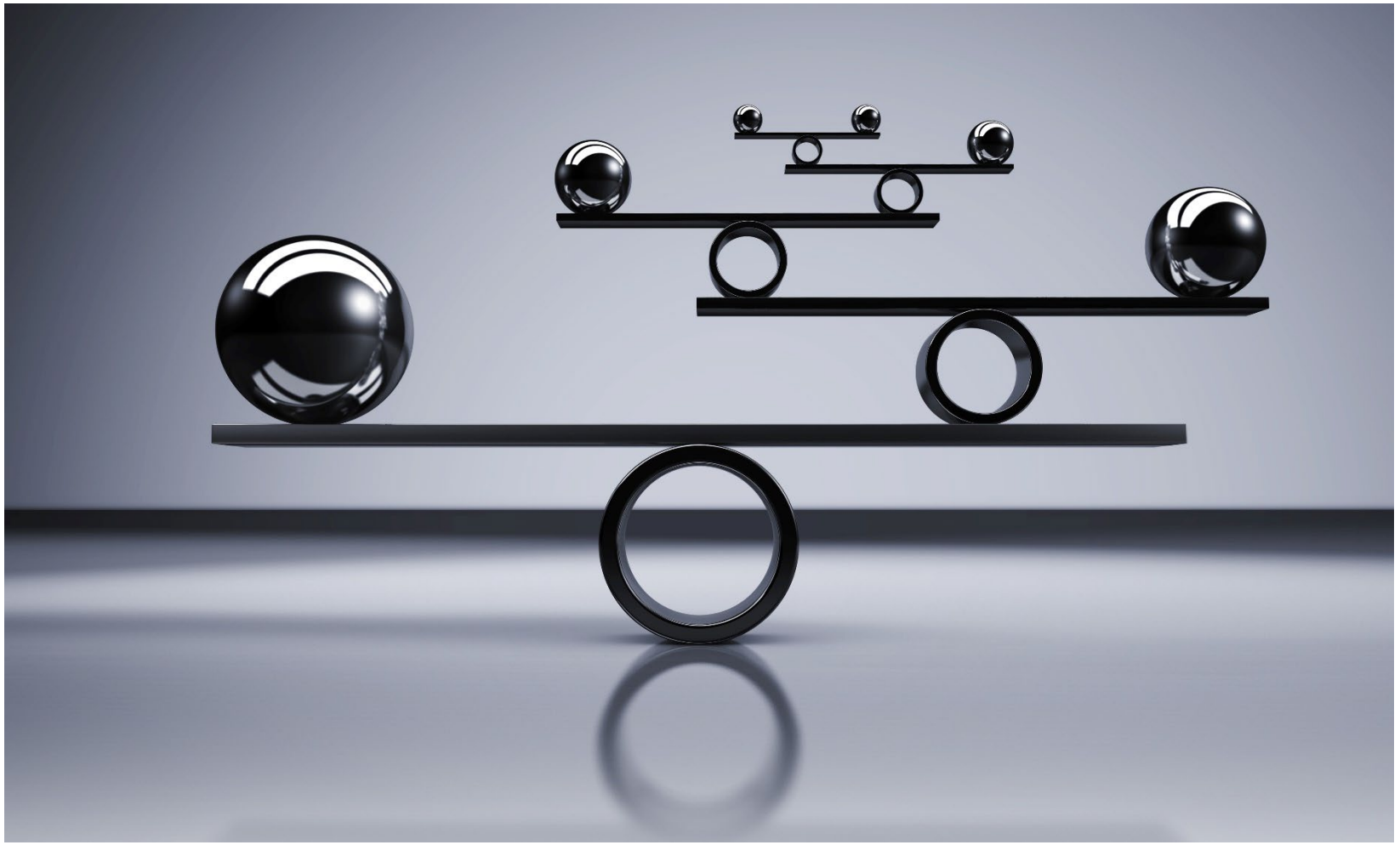
Increasing Attention on Reproducibility for Your Own Sake

- Supercomputer cycles are scarce resources
- No one wants to spend their precious allocation running simulations two or three times to be confident of the results
  - Though this ends up happening more than most people admit
  - And it could still be wrong!
- But lots of people need to have confidence in your results
  - You
  - Your project lead or boss
  - Your sponsor
  - Your reviewers or referees
  - Your readers
- Need to think about how to build credibility *without* repeating runs





How to Improve Reproducibility



Strategies for Improving Reproducibility *During* Development (1/3)

- **Solid versioning practices are fundamental to reproducibility**
- Version control of code, documentation, and other artifacts
  - Frequent commits (perhaps to a separate development branch)
- Provide versioning information in key output(s)
  - Version numbers (i.e., semantic versioning) are useful, but when do you increment them?
  - Automatic identifiers (i.e., git commit hash) are less ambiguous, but may not be as meaningful
  - Is the code you're building modified from the version in the repository? *(Not often done in practice)*
- Maintaining documentation (and other artifacts) in sync with code
  - You'll forget
  - Or you won't have (make) time to go back to it

Strategies for Improving Reproducibility *During* Development (2/3)

- **Build in quality from the start**
- Define and follow coding standards
  - Not just code style
  - Expectations for kinds and extent of documentation, types and rigor of tests
- Develop tests as you code
  - Write tests while the code is fresh in your mind
  - Test Driven Development (TDD) means write tests before code, then code to pass the tests
- Require increasingly rigorous testing as the code becomes more "public"
  - Testing has costs, need to balance level of risk against cost of creating and executing tests
  - Also think about frequency of tests at different levels of cost (c.f. continuous integration)
- Practice peer code review
  - Per commit – should meet standards, *and* be understood and judged correct by reviewer
  - Pair experienced reviewers with less experienced coders to help ensure quality
  - Retrospective if you have a lot of existing unreviewed code

Strategies for Improving Reproducibility *During* Development (3/3)

- **Understand the numerics of your code**
- Floating point numbers are just approximations to real numbers
  - Many numerical methods have “quirks” too
- If you’re using reduced- or mixed-precision computations, carefully compare with full-precision versions
  - On paper during development of the algorithms
  - Maybe provide an alternative full-precision computational path
- Consider the possible effects of non-determinism due to concurrency
  - Floating point calculations done in different order may yield different results
  - Maybe useful to have an option to force deterministic computation
  - Look for testing/verification methods that don’t depend on bitwise reproducibility
- Know your error bounds and develop tests against them
  - E.g., conservation rules apply to many physical quantities
- Consider consulting subject matter experts for help

Strategies for Improving Reproducibility *After* Development

- **Testing, testing, and more testing!**
- Add "regression tests"
  - If you fix a bug, add a test to make sure that bug doesn't creep back in
- Add more tests
  - Be creative
  - Think about common cases, then corner cases
  - Think about misuse (unintentional or intentional)
  - Think about synthetic tests with synthetic data
  - Think about low-cost tests that can be "always on" (even if they're not so stringent)
  - Can you detect silent data corruption?
- Test your third-party dependencies
  - Are your tools doing what you think they're doing?
  - What if you're using a new version?
  - How do you decide if it is okay to upgrade to a new version?
- Test your tests!
  - Make sure tests fail when they're supposed to!
- Thoroughly verify the code
  - Does the code do what you intended it to do?
  - On all relevant platforms (compilers, hardware, etc.)
- Test regularly
  - To identify and document where changes to the underlying platform change code behavior/results

Digression – “Physics” (or Math)-Based Testing Strategies

- **Use what you know (or can construct) about the model you’re studying to test its implementation**
- Synthetic operators with known properties
  - Spectrum (huge diagonals)
  - Rank (by construction)
- Invariance principles
  - Translational, rotational, etc.
  - Physical symmetries
  - Mathematical symmetries
- Conservation rules
  - Fluxes, energy, mass, etc.
- …

Digression – Design by Contract Programming

- **Building testing into your routines**
  - **To complement, *not replace*, other testing**
- The interface to a routine can be thought of as a contract between *caller* and the *callee* (the routine)
  - What does the routine expect on input? **preconditions**
  - What does the routine guarantee at completion? **postconditions**
  - What does the routine leave unchanged? **invariants**
- Given valid inputs (preconditions satisfied) a routine should guarantee valid outputs (postconditions satisfied, invariants maintained)
  - If the preconditions are not satisfied, the routine should return an error
  - Emphasize low-costs tests that can be always-on
  - May need to be able to switch enforcement of expensive tests on/off (but try not to!)
- Making the contract explicit facilitates correct use of routines
  - Especially when routine is reused in another context
  - Especially by those not intimately familiar with them

Strategies for Improving Reproducibility by *Planning* Experiments

- Plan your experiments thoroughly
  - If you're in a team, designate *one* person to coordinate the experimental campaign
  - Know what you need (in the code, as inputs, as outputs to capture/analyze, etc.)
  - Know what to expect (in results, performance/cost, etc.)
  - How will you convince yourself that your results are trustworthy?
- Develop helpful diagnostics
  - Low overhead ways of confirming the health of the run
    - Are conserved quantities conserved?
    - Has any quantity become unphysical?
- Develop hierarchy of analysis
  - Full analysis of runs is often not feasible while simulations are running
  - Intermediate level analysis can give further insight into health of the simulation
- Perform pilot/test runs to build confidence in correctness, performance, scaling
  - Often useful to pursue an incremental/layered strategy
- Ensure that you have the resources to store and/or analyze the outputs
  - What can you afford to archive?
  - What will you need to process and delete?
  - What will you need to process during execution or stream?

Consider: use a lab notebook to keep track of all this planning: motivation, reasoning and decisions, consequences (more to follow)

Strategies for Improving Reproducibility *During* Experiments

- **Can you reproduce the code used for each and every experiment?**
  - **Three years later?**
- Use only well-defined versions of code (i.e., official “releases”, tags, etc.)
  - Separate ongoing development from science campaigns using separate branches
    - Main or development branches are often moving targets
  - Capture the exact version of the code used for each experiment
    - Is the code you’re building exactly what’s in the version control repo?
  - Don’t change versions during a related series of experiments (unless you have to)
    - Make changes only to the science branch
    - Reconcile the science branch changes with the development branch only *after* the science is done
  - If you have to change versions, know exactly what changed
    - Capture the exact version of the code used for *each experiment*
- Use only versions of code that have been thoroughly verified
- Continue to use regular testing to identify changes due to the underlying platform
  - E.g., compiler release introduces a new optimization that changes numerical results
- Consider capturing version information of key libraries, compilers, and other dependencies used to build code
  - *Not often done, in practice*
- *Put this info into your lab notebook too!*

Conversations with Carlo

Carlo Graziani is a Computational Scientist at ANL
BSSw blog article *HPC and the Lab Manager*

- As researchers' careers progress
  - The problems become more complex and larger
  - Previous informal techniques for executing a study start to fail
  - The researchers sense that something is missing
  - They invent processes and tools to compensate

This happened to Carlo and at some point, he realized that

"I had re-invented the lab notebook!"

A minimal definition of a lab notebook

A goal of keeping a lab notebook is "...to write with enough detail and clarity that another scientist could pick up the notebook at some time in the future, repeat the work based on the written descriptions, and make the same observations that were originally recorded. If this guideline is followed, even the original author will be able to understand the notes when looking back on them after considerable time has passed!"

- Howard Kanare, *Writing the Laboratory Notebook* *(emphasis added)*

Lab notebooks are…

- A fundamental part of communication as well as rigorous, reproducible science in a lab,
- A common-place or required part of an experimental laboratory,
- Populating a scientific “lab notebook” was an “automated” process at the observatory,
- A tool for preventing scientific fraud, and
- Record of invention and defending against allegations of fraud.

Lab notebooks

- Should be used regularly,
- Should be comprehensive and never filtered,
- Don’t need perfect grammar and full sentences,
- Content is frozen at creation, and
- Hopefully contains more than just data (*e.g.*, motivation, reasoning, conclusions).

They focus on data and information more than knowledge and understanding, **but** people interact, evolve, and grow by collaborating through the notebook.

Example notebook entries

A bad example

Monday July 25, 2022
9:05 am - Do study ABC
8:47 pm - Lot's of interesting data!
- Results are in GCE

A better example

Monday July 25, 2022 (Jared)
9:05 am - Continuing work for study ABC. (See July 7)
- I presently believe that if A happens, then B must also happen.
- To verify this, I intend to
  - …
  - ***

9:30 am - Started executing this experiment on Bebop.
- Built debug version of binary with Intel 20.4
- Based on clean commit 5a43b21c
- Build log saved to `my_test_2022.log`
- No errors or warnings emitted
- Used job script `run_my_test` with configuration 24 (Job ID 123456)
- Stdout/err & results saved in folder ABC

10:07 am - Analysis run with script `analyze_my_test.py` and results saved in same folder.
- Since no peak seen around 1.5 MeV, I was wrong. But based on this, I now *believe* that if A happens, then C must also happen.

No one likes writing lab notes

“Lab notes are a waste of time. I write notes but never use them.”

- Almost all newcomers to lab notebooks

- This attitude comes from lack of experience & awareness
  - Good notes are implicit communication & sharing
  - Good notes can be turned into procedures
- Lab notes become more useful as time passes
  - Our memory fades
  - It can take years before we see the benefit
  - The beneficiary is often our “future self”
- Writing but not reading lab notes is a good thing
  - Lab notes are most useful when something has gone wrong

Criteria for lab notebooks for computing?

- **Paper won't work.** We work anywhere and sometimes in distributed way.
- Should notebooks be public and how to do that?
- How many different types of notebooks do we need?
- Do we use a single electronic lab notebook (ELN) or distribute notes across a suite of tools?
- How can we use automation appropriately to overcome difficulties and increase productivity?

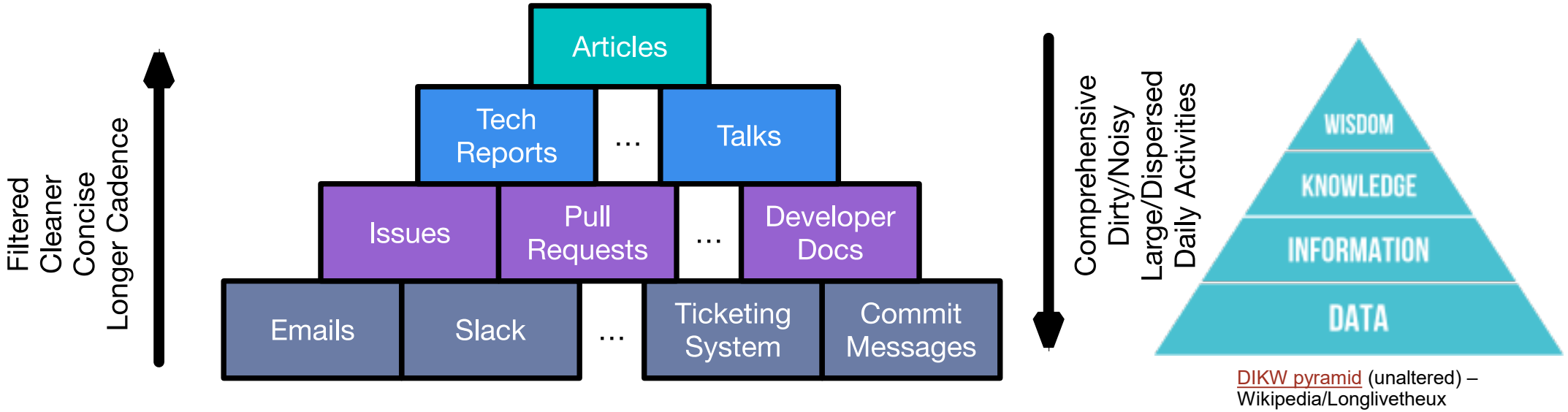
We likely need multiple streams of lab notes

Different streams of lab notes

- Lab notebook for changes to scientific instrument
  - Changes in code repo necessary for study
  - Changes to SW environments
  - Changes to build/job files and build systems
- Lab notebook to detail how experiment was designed and executed
- Lab notebook for data analysis tools
- Right tool for the job
  - We don’t want a single 10,000-line README

Which leads us to documentation

Documentation is knowledge communication & can build understanding



DIKW pyramid (unaltered) – Wikipedia/Longlivetheux

- Data at the bottom. Knowledge as we move up?
- Some documents frozen at creation. Others are living.
- Does this capture how hard it is to do documentation in a distributed, digital world?

Git lab notes stream

Keep lab notes for your software as close to the “instrument” as possible

```
Date:   Mon Jun 27 10:42:22 2022 -0500

    (Issue #215) Added in Milhoja Init unit test.  I have tried to structure this in
    accord with the unitTest architecture in the User Manual.  One main consequence
    of this is that it uses the generic Grid unitTest evolveAll routine, which
    writes a unitTest output file.  Since this test also uses the ut_testDriverMod
    framework, I had to simplify that so that it doesn't attempt to write the same
    file.  This change will likely affect the AMReX unit tests.

    I have run this successfully in 1D, 2D, and 3D on GCE/compute-12 with Intel.
    The test correctly creates a unitTest_XXXX output file and adds in a success
    line only if the test was 100% successful.  I temporarily dumped the ICs and
    final solution to AMReX-format files and manually confirmed correct content.
```

Details not obvious from commit diff:
Motivation, reasoning, consequences

Testing notes

*This message is missing a title as the first line.

This is only one component of the lab notebook
for software – combine with pull requests

Pull request as a “filtered” notebook entry

- A pull request is an aggregation of commits to a git repo
  - Individual commits are linked to the pull request
- The PR allows for additional content that’s distinct from the individual commits
- Use PR description to record process to verify correctness of changes
  - 2-2.5 days effort carried out over a week
  - Copy/pasted from previous PR and adapted first (designed process)
  - Improved as carried out process – converging on a quasi-procedure
  - Filtered so that reviewers aren’t overwhelmed
  - Helped organize effort & design good tests
- Additional benefits
  - Senior reviewers provide feedback & suggest improvements
  - Junior reviewers exposed to work habits of other people
- Example: Flash-X PR #247 on next slide

Example:
Flash-X
PR #247

jared321 commented on Aug 18 • edited ▾ Author ☺ •••

Working on GCE/compute-012 with

```
Currently Loaded Modules:
  1) intel/20.4   2) mpich/3.4.2-intel   3) hdf5/1.12.1-mpich-3.4.2-parallel-fortran
```

- All tests will be built in debug mode using the Intel/20.4 + MPICH + HDF5 stack loaded via GCE modules.
- The same pseudo-UG-only testing was carried out identically for the 2D/Sedov/simpleUnsplit and 3D/Sedov/Unsplit simulations with Paramesh, AMReX, and Milhoja. These two simulations were chosen as they will hopefully become official Milhoja Comparison tests in the GCE test suite.
- All 2D tests ran in 304 steps with 6 checkpoint files including the initial conditions; 3D, 61 steps and 2 checkpoint files.
- Ran the 2D tests with 4 MPI processes; the 3D, with 8 MPI processes. These are the number of processes presently used for Sedov tests in the GCE test suite.
- Confirmed with `sfocu` that the Paramesh and AMReX final checkpoints are identical and that the Paramesh and Milhoja final checkpoints are identical.
- Confirmed that the integrated quantities were conserved in each case up to a reasonable level and that the initial values were consistent across all associated runs to at least 12 decimal digits.
  - standard deviation of 2D conserved quantities less than 2.5e-14
  - standard deviation of 3D conserved quantities less than 3.25e-14
  - Confirmed that the initial values of the [xyz]-momenta were all exactly zero and that the z-momenta was exactly zero for each all time steps in the 2D simulations.
- Confirmed in Milhoja 2D case that we get, as expected, the identical final checkpoints if we run with 4 and 8 MPI processes.
- Where possible, the `milhoja.log` files were reviewed to confirm correct configuration and execution.
- Viewed the Milhoja final checkpoints in Visit to qualitatively confirm reasonable results for all variables and correct mesh overlays. 3D data was viewed with a slice at 50%.

README lab notes streams

- High-level road maps with motivation, documented decisions, and conclusions
  - Concise living docs that function as executive summaries
  - Higher up in documentation hierarchy
- Low-level notes such as managing software stack
  - Traceability of SW environment & therefore verification
  - Can be turned into procedures

High-level README

My Study Executive Summary

This repository encapsulates the computational laboratory environment constructed and used for our 2022 My Study scientific research. Not only does it contain all tools needed to setup and use the environment, but it also contains the metadata and context needed to understand how data was acquired and how to use it for analysis and drawing conclusions.

Study Goals

Our motivation for this study is ...

We believe that ...

We intend to use our software to ...

To accomplish our goals we require that ...

We understand that our software is limited and therefore that our study is limited in the sense that ...

Some optional goals are ...

Ideally, we would like to publish our results in the Journal ZYX with a submission target date of ...

Organization

ABC will be the PI and will ...

XYZ is responsible for ...

Low-level README

AMReX Installation History

5 March 2020 - FlashFluxRegister b1dd083

- Using changes from FlashFluxRegister branch by Weiqun.
- Need for improved flux handling across all branches.
- Built 1D/2D/3D libraries for both gfortran and Intel at this commit.
- These were tested with the Staged/Staged-Intel/Master compute00x test suites with no other changes made. All tests passed.

6 November 2019 – master – 93fb085d

- Austin encountered an AMRex based bug and reported it in Issue #643. He reports that AMRex fixed that bug at commit - 23f943f.
- New install location is /nfs/proj-flash5/ a shared group at MCS, we are no longer going to use the /sandbox/flash/ versions on compute00[123]
- Built 1D/2D/3D libraries for both gfortran and Intel at this commit.
- These were tested with the Staged/Staged-Intel/Master compute00x test suites with no other changes made. All tests passed.

29 March 2019 – development – 06c6c0e2

- Rebuilt the libraries from a commit that contains the multifab changes implemented manually in the previous libraries. These changes were merged in to AMReX at commit 66392f8d on Tue Mar 26. No code in AMReX needed modifying for this build.
- These libraries were also built with Particles and linear solvers enabled for Saurabh's tests. He showed me that I had to update the automagically updated makefile to get the Particles Fortran interface into the libraries.
- Built 1D/2D/3D libraries for both gfortran and Intel at this commit.
- These were tested with the Staged/Staged-Intel/Master compute00x test suites with no other changes made. All tests passed.

20 March 2019 – master – 884d3194 (modified)

- Updated the files AMReX_multifab_fi.cpp and AMReX_multifab_mod.F90 at commit to incorporate the local tile index in the Fortran interface. These changes had already been tested during development in local repositories.
- Built 1D/2D/3D libraries with both gfortran and intel on compute001. These were then copied over to compute002 and compute003.

Capturing data context & metadata (distinct from the data)

- Build dates, user, system name, git hashes, configuration data in file headers
  - Self-documenting files
- Build & job logs
  - Software environment info (e.g., modules, `ldd` output)
  - git diffs
  - Environment variables
- Automate as much as possible

Jupyter notebooks

- Jupyter notebook can put context & metadata next to data
  - High-level design & motivation up top
  - Low-level lab notes for acquiring data
  - Load & use data
  - Generate visualizations in place
  - Comment on results
- Where do notebooks fit into the documentation hierarchy?
- Repetitive use of notebook?
  - Limit amount of code in notebook

How to organize your “virtual” (multi-stream) lab notebook?

- It should be
  - Easy to create and maintain lab notes
  - Easy to concentrate more on executing work & less on documenting it
  - Easy to find what you need
- Each stream should
  - Have a clear identity for what it records
  - Not contain notes contained in other streams
  - Be recorded by using the right tool for the job

Strategies for Improving Reproducibility *with Lab Notebooks*

- **Be thorough in capturing provenance**
  - **Agents (codes), entities (inputs, outputs, etc.), activities (the transformation)**
- Capture code version
- Capture all inputs/configuration information for each experiment
- Use multiple systems to ensure that you can correctly associate inputs, outputs, and code versions
  - Systematic directory and file naming conventions (document them!)
  - Separate written notes (paper notebook, electronic notebook)
    - Lab notebooks aren't just for people who literally work in a laboratory!
  - Scripts to orchestrate experiments (versioned and captured)
  - Version control (if data is not too large)
- Capture important outputs (as feasible)

Strategies for Improving Reproducibility *After* Experiments

- **Continue provenance capture through data analysis/reduction process**
  - **Agents (codes), entities (inputs, outputs, etc.), activities (the transformation)**
- Script as much of your analysis/reduction as possible
  - Prefer scriptable tools over those requiring human interaction
  - Keep the scripts under version control
- Document your process thoroughly
  - Separately from scripts, etc.
  - E.g., in your lab notebook
  - Especially where human interaction is required
- Capture key intermediates in the reduction process
  - The longer the processing pipelines, the more likely problems will creep in
  - The more you capture, the more you will have for verification (and to find problems) later
- Capture the data (in machine-readable form) used to produce graphs and tables
  - Expected by basic data management plans
  - And an increasing number of publishers

Tools May Help with Reproducibility

Just a small sampling…

- Containers to capture the software
- Resources for finding, understanding, and debugging floating point math problems: http://fpanalysistools.org/
- Cloud platforms to publish and reproduce research code and data
  - E.g., https://CodeOcean.com
- DOIs and hosting of data, code, documents, etc.
  - E.g., https://zenodo.org/, https://FigShare.com

Make sure to test and understand your tools thoroughly *before* using them for something important!

Summary

- The credibility of your science derives from the credibility of your code (and process)
- Science stakeholders are ratcheting up expectations for reproducibility
- Reproducible results are a necessity, not a luxury
  - There is no credible science without full provenance
- Computational science campaigns can be expensive in time and resources
  - Care and planning is vital to ensure that outcomes meet expectations
- There are strategies to improve reproducibility in all phases of the scientific process
  - During development
  - After development
  - During experiments
  - After experiments
- They amount to better software development practices
  - The same kinds of practices advocated for reasons of productivity, sustainability, maintainability, etc.

Other resources

- The FAIR Guiding Principles for Scientific Data Management and Stewardship. Mark D. Wilkinson, et al. https://doi.org/10.1038/sdata.2016.18
- FAIR for Research Software (FAIR4RS) Working Group: https://www.rd-alliance.org/groups/fair-for-research-software-fair4rs-wg/forum/
- Editorial: ACM TOMS Replicated Computational Results Initiative. Michael A. Heroux. 2015. *ACM Trans. Math. Softw.* 41, 3, Article 13 (June 2015), 5 pages. DOI: http://dx.doi.org/10.1145/2743015
- Enhancing Reproducibility for Computational Methods. Victoria Stodden, Marcia McNutt, David H. Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A. Heroux, John P.A. Ioannidis, Michela Taufer Science (09 Dec 2016), pp. 1240-1241. DOI: 10.1126/science.aah6168
- Simple experiments in reproducibility and technical trust by Mike Heroux and students (work in progress), https://betterscientificsoftware.github.io/Trust-Tools/
- What every scientist should know about floating-point arithmetic. David Goldberg. https://doi.org/10.1145/103162.103163
- Carlo Graziani, *HPC and the Lab Manager*. **Better Scientific Software**. https://bssw.io/blog_posts/hpc-and-the-lab-manager. Nov 17, 2021.
- Howard M. Kanare, *Writing the Laboratory Notebook*. American Chemical Society, Washington, D.C., 1985.
- Nicole Brewer, *Jupyter4Science: Better Practices for Using Jupyter Notebooks for Science*. **Better Scientific Software**. https://bssw.io/items/jupyter4science-better-practices-for-using-jupyter-notebooks-for-science. March 17, 2024.