A Workflow for Increasing the Quality of Scientific Software (in Computational Science and Engineering)

IDEAS Productivity Project Webinar 2021-04-07
Motivation: multiphase flow simulation software

- Fluids that do not mix are separated by an interface $\Sigma(t)$ (surface in 3D).
- Goal: track $\Sigma(t)$ as it moves in time $t$ and changes its topology.
LEIA methods\(^1,2,3\) require thorough testing:

- Verification cases: evolution of \(\Sigma(t)\) and two-phase flows with exact solutions.
- Validation with respect to experiments.
- Serial and parallel computational efficiency.

---


Publish or perish \(^4\) prioritizes publications over scientific software.

Dedicated resources for increasing software quality are usually not available.

Ph.D. students rotate every 4-5 years, postdocs every 1-2 years.
  - Little or no overlap between successors and predecessors.

Large-scale software design is not a necessary part of the CSE curriculum.
  - Different CSE background: (Applied) Mathematics, Mechanical Engineering, Physics, Informatics.

Real-world example: onboarding people into \(\text{OpenFOAM}\) module development.

\(^4\) Symbol of a publish-or-perish simplification of the workflow :)}
Computational Science and Engineering software in university research groups

Divergence

- Not being able to continue development from an earlier state.
- Reproducing results from a publication is not possible.
  - Data, source code and publication are not archived and cross-linked.
  - The version used to generate the data is not documented.
- Not being able to re-use a model from a publication.
  - The model is not implemented in a modular way.
  - Version integration was not done.
  - Non-granular commits were used.
- Having no overview of the impact of a change on the rest of the module.
A workflow for increasing the quality of scientific CSE software

1. Track the issues in a Kanban board.
   - Model issues as Progress Tracking Cards⁵.

2. Use a simple version-control branching model.

3. Apply Test-Driven Development (TDD) for CSE software.

4. Enable Continuous Integration with an emphasis on result visualization.

5. Cross-link software, result data, and report/article when reaching a milestone.
   - When submitting a publication to peer-review.
   - After the publication has been accepted.
   - When giving up on an idea.

6. Bonus step: publish a Singularity image with the code and data.

⁵Developed by Better Scientific Software.
The workflow is developed with OpenFOAM projects but it is tested with other software.

**Disclaimer:** This offering is not approved or endorsed by OpenCFD Limited, producer and distributor of the OpenFOAM software via www.openfoam.com, and owner of the OPENFOAM® and OpenCFD® trade marks.
University research teams working on the same project are generally small (2 - 5 members).

Separation of Concerns (SC) and Single Responsibility Principle (SRP) significantly simplify the branching model.

Separation of Concerns: code is organized in non-overlapping layers and sections.

Single Responsibility: functions or classes perform single clear tasks.

SC and SRP can be applied to any software.

Dogmatism should be avoided: single responsibility vs less responsibilities.

OpenFOAM already uses object-oriented and generic software design patterns.
Maintainers (postdocs, experienced Ph.D. students) manage the integration.

- Keep the branching model as simple as possible.
- Main and development branches are protected and managed by Maintainers.
- Maintainers are responsible for git tags and cleanup:
  - **Main**: integrations from *accepted publications* and *development branch*.
  - **Development**: integration of *(CI)-tested improvements*.
  - **Feature**: SRP reduces git-conflicts with researchers working on different files.
- Complex branching workflow ⇒ complications with onboarding new members.
TDD$^6$ for CSE

- Define verification and validation tests at the start.
- Focus placed the final result: interpolation, integration, discretization, PDE solution, physics.
- Top-down, instead of bottom-up test coverage.
- Don’t go overboard with unit-tests $\hat{\circ}$: extend unit-tests when debugging a failing CSE test.
- Focus kept on tests with real-world (publication) input.

---

Test Driven Development
Verification and validation tests define the Application Programming Interface

- **New code**: it is easier to program the API you wish for, if you are its first user.
  - Make the class interface easy to use correctly and difficult to use incorrectly\(^7\).
  - Reduce number of function arguments, single responsibility, clear naming, ...

- **Legacy code**: extend existing API without modification.
  - OpenFOAM: understanding class hierarchies, *finding a base class with Runtime Type Selection and a virtual function to overload*.

- **The test application is the solver application with a different input**.
  - If possible, testing and solution is done by the same code.
  - This prevents code duplication.
  - Data output and additional checks can be disabled by (compile-time) options.

---

\(^7\) Scott Meyers. 2014. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14 (1st. ed.). O'Reilly Media, Inc.
Jupyter notebooks

- **Documentation**: geometry, initial and boundary conditions, error norms, comparison data.
- **Processing**: verification errors (conservation, convergence, stability), validation errors.
- **Result analysis**: very straightforward, interactive, remote.

---

[8] https://jupyter.org/
Test Driven Development
Parameter tests

A Workflow for Increasing the Quality of Scientific Software (in Computational Science and Engineering) -
T. Marić, JP. Lehr, I. Pappagianidis, B. Lambie, D. Bothe, C. Bischof
The quality of CSE software is measured using verification and validation data.
Effective comparison with others (previous versions) hinges on data organization.

**Legacy code:**
- use the existing folder structure and parameterization tools
- The mapping (case000) → (parameter vector) must be stored (YAML, ...)

**New code:**
1. Simple folder and file structure
2. HDF5 or other open data format.
3. Alternative to HDF5: ExDir

---

9 [https://www.hdfgroup.org/solutions/hdf5](https://www.hdfgroup.org/solutions/hdf5)
pandas.MultiIndex CSV with metadata for secondary data

- pandas.MultiIndex saved in “metadata columns”.
- Metadata is repeated: not an issue for the small secondary data!
- Metadata in columns → pandas.MultiIndex → strongly simplified data analysis.
- Direct readable export of tables to LaTeX!

<table>
<thead>
<tr>
<th>VELOCITY_MODEL</th>
<th>H</th>
<th>L_INF</th>
<th>O(L_INF)</th>
<th>EPSILON_R_EXACT_MAX</th>
<th>O(EPSILON_R_EXACT_MAX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHEAR_2D</td>
<td>0.125000</td>
<td>0.032961</td>
<td>1.833407</td>
<td>0.032961</td>
<td>1.833407</td>
</tr>
<tr>
<td>SHEAR_2D</td>
<td>0.062500</td>
<td>0.009249</td>
<td>1.955529</td>
<td>0.009249</td>
<td>1.955529</td>
</tr>
<tr>
<td>SHEAR_2D</td>
<td>0.031250</td>
<td>0.002385</td>
<td>1.988745</td>
<td>0.002385</td>
<td>1.988745</td>
</tr>
<tr>
<td>SHEAR_2D</td>
<td>0.015625</td>
<td>0.000601</td>
<td>1.997178</td>
<td>0.000601</td>
<td>1.997178</td>
</tr>
<tr>
<td>SHEAR_2D</td>
<td>0.007813</td>
<td>0.000150</td>
<td>1.999294</td>
<td>0.000150</td>
<td>1.999294</td>
</tr>
<tr>
<td>SHEAR_2D</td>
<td>0.003906</td>
<td>0.000038</td>
<td>1.999294</td>
<td>0.000038</td>
<td>1.999294</td>
</tr>
</tbody>
</table>
Continuous Integration with result visualization

Schematic diagram

- User
  - Push changes
  - Merge request
  - Process and visualize results
  - Run Tests
  - Accept merge request

- User Repository
  - Triggers CI Pipeline

- Main Repository
  - GitLab Runner

- GitLab CI Pipeline
  - CMake
  - Python
  - Markdown

- Publish results
  - Process and visualize results
  - Run Tests
  - Build

- Yes
  - Tests passed?
  - Published on
  - Accept merge request

- No

A Workflow for Increasing the Quality of Scientific Software (in Computational Science and Engineering) - T. Marić, JP. Lehr, I. Pappagianidis, B. Lambie, D. Bothe, C. Bischof
1. **Short few CPU-core tests**: work-PC.

2. **Short many-core tests**: obtain a workstation with a 64-Core CPU.

3. **HPC tests**: combine 1. or 2. with an HPC cluster.

An HPC cluster is relevant for production tests and performance measurements.

- This workflow uses coarse (“smoke”) tests
  - Unit tests run for 1. and 2.
  - Convergence ensured for 1. and 2.
  - Is efficient in parallel for 1. and 2.

- **Challenge**: Is it possible to combine 1., 2. and 3. and publish instead of perish?

---

\(^{11}\)Thanks to CRC 1194 at TU Darmstadt.
Build a Docker image for your software, and track the Dockerfile with the project.

**Example OpenFOAM Dockerfile** on `ubuntu:focal` with "system" open-mpi and scotch.

On the testing machine

- Install Docker and GitLab runner and register the GitLab runner with a Docker executor.
- Configure the GitLab runner in `/etc/gitlab-runner/config.toml` to
  - use a local Docker image, e.g., `image = "openfoam-v2012Ubuntu-focal:latest"`, and
  - never pull images `pull_policy = never`. 
Files created within a job are gone when the job ends.

GitLab uses job artifacts to pass on data from one job to the next.

Job artifacts only work with files stored in project's sub-folders.

Libraries and applications are passed to other jobs as artifacts.

Artifacts can be downloaded on the GitLab project website.
Out-of-source installation: binaries only available outside the repo!

- Use environment variables to build and pass on artifacts
  - `$FOAM_USER_LIBBIN` folder stores library binaries.
  - `$FOAM_USER_APPBIN` folder stores application binaries.

Build job:
- create artifact folders inside the repo,
- copy library and application binaries to artifact folders,
- export artifact folders.

Run job: simplified copying of binary artifacts to OpenFOAM folders
- `mkdir -p {FOAM_USER_LIBBIN, FOAM_USER_APPBIN}`
- `cp FOAM_USER_LIBBIN/* FOAM_USER_LIBBIN`
- `cp FOAM_USER_APPBIN/* FOAM_USER_APPBIN`
- Run tests.
Continuous Integration with result visualization

Schematic diagram

- User pushes changes to User Repository.
- Merge request is sent to Main Repository, which triggers the CI pipeline.
- The CI pipeline runs tests and processes the results.
- If tests pass, the results are published to GitLab Pages.
- Users can accept merge requests.

**Tools and Languages**

- GitLab Runner
- GitLab CI Pipeline
- Python
- CMake
- Markdown
- GitLab Pages

**Published on**

- GitLab Pages

**Runs on**

- GitLab Runner

---

A Workflow for Increasing the Quality of Scientific Software (in Computational Science and Engineering) - T. Marić, J.P. Lehr, I. Pappagianidis, B. Lambie, D. Bothe, C. Bischof

IDEAS Productivity Project Webinar 2021-04-07
jupyter nbconvert notebook.ipynb --execute --to FORMAT

- Execute each jupyter notebook in the repository.
- Notebooks agglomerate secondary data into pandas.MultiIndex CSV files.
- Export secondary data and notebooks in different formats as artifacts.

**Visualization**
- Download the artifact and open the notebook 📝.
- **Alternative:** publish the notebook as a blog post in a GitLab Static Page project.
- Notebooks contain information on failing tests.
- Mapping "caseXYZ" → "parameter vector" is crucial for re-starting failed parameter variations!
Continuous Integration with result visualization

Test evaluation

Very straightforward

- Python scripts test secondary data agglomerated by notebooks from simulation results.

- **Examples:**
  - Is the order of convergence of an error norm $\geq 2.0$?
  - Is is the difference between simulation and experiment data $\leq 4\%$?
Example OpenFOAM CI project
Cross-linking data, source code and reports/publications

Schematic diagram

A Workflow for Increasing the Quality of Scientific Software (in Computational Science and Engineering) -
T. Marić, JP. Lehr, I. Pappagianidis, B. Lambie, D. Bothe, C. Bischof
Cross-linking data, source code and reports/publications

Singularity

- Whence the Singularity Image\(^{12}\)?
  - More intuitive than Docker: **Singularity handles images as files**.
  - Built for HPC from the start.
  - Doesn’t require root rights.
  - Results as actual files, not ”data in spinning containers”.
  - Maps user folder to the container: result data remains on the host.

- Why not replace Docker with Singularity within GitLab CI?
  - We’re learning how to do this using **GitLab custom executors**.
  - Does the workflow still survive publish-or-perish test?

- Why a source-code snapshot on-top of the image and the repository?
  - Repositories get migrated, deleted, and some researchers still fear images.
  - Quick and direct access to source code from the publication.

\(^{12}\)https://sylabs.io/docs/
The source code and the data stored in the image can be quickly reproduced.

Article reviewers can clone, build, run and visualize easily.

Example: Singularity Image from an active review

- Clone the code repository from the image:
  `geophase-JCOMP-D-19-01329R2.sif clone geophase`

- Build:
  `geophase-JCOMP-D-19-01329R2.sif build geophase build`

- Run tests:
  `geophase-JCOMP-D-19-01329R2.sif run-tests geophase build`

- Open the jupyter notebook:
  `geophase-JCOMP-D-19-01329R2.sif jupyter-notebook geophase`
Similarity with other workflows / best practices

Our *(subjective)* estimates* of similarity 1 – 5 (higher is more similar), −: aspect not addressed.

<table>
<thead>
<tr>
<th>DOI</th>
<th>Branching model</th>
<th>TDD</th>
<th>Cross-linking</th>
<th>CI</th>
<th>(Meta)data standardization</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.12688/f1000research.11407.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>10.3934/math.2016.3.261</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>10.1371/journal.pbio.1001745</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10.1371/journal.pcbi.1005510</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>10.1145/2723872.2723881</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>10.1145/3324989.3325719</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>10.1371/journal.pone.0230557</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>10.1145/3219104.3219147</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>

*The list may still be incomplete.*
Lessons learned

- Keeping the workflow as simple as possible is crucial for acceptance.
- Focusing on secondary data simplifies the workflow significantly.
- For simulations that run < 24 hours primary data can be recomputed easily.
- Periodical cross-linking of research data is done quickly and it is very beneficial.
- Personal responsibility is crucial at University research groups: who are the maintainers?
  - What are the incentives for maintainers?
- Fixing the (parallel) I/O of legacy scientific codes requires a large amount of effort.
  - It should be done outside of research projects.
Outlook

- Performance CI jobs run on 64-core workstations: moving on to the HPC cluster.
- Singularity GitLab executor?
- Jupyter Hub for interactive analysis of problems in parameter variations?
- Automatic publishing and cross-linking of CI artifacts?
  - Source code archive, Singularity container, secondary data.
  - Data repository API must be used to modify metadata.
Acknowledgements

Funded by the German Research Foundation (DFG) – Project-ID 265191195 – CRC 1194 : Z-INF