Quality and Reliability in CFD

Open Source Challenges

Hrvoje Jasak

Wikki Ltd, United Kingdom

Faculty of Mechanical Engineering and Naval Architecture

University of Zagreb, Croatia

Outline

Objective

- Describe how object-oriented software design and layered architecture impacts the quality and reliability of the software
- Review validation and verification practices in place with FOAM/OpenFOAM and in the framework of the FOAM-Extend project
- Summarise the quality assurance problems related to open source software development model

Topics

- Introduction: OpenFOAM and object-oriented CFD software design
- Code verification: layered development and generic programming
- Code validation framework: tutorials and best practice guidelines
- Example: validation of the naval hydrodynamics solver
- Summary and Outlook

OpenFOAM: Executive Overview

What is OpenFOAM?

- **OpenFOAM** is a free-to-use Open Source numerical simulation software with extensive CFD and multi-physics capabilities
- Free-to-use means using the software without paying for license and support, including **massively parallel computers**: free multi-1000-CPU CFD (8192)
- Software under active development, capabilities mirror those of commercial CFD
- Substantial installed user base in industry, academia and research labs
- Numerous extensions to non-traditional, complex or coupled physics

Main Components

- Discretisation: Polyhedral Finite Volume Method, second order in space and time
- Lagrangian particle tracking, Finite Area Method (2-D FVM on curved surface)
- Massive parallelism in domain decomposition mode
- Automatic mesh motion (FEM), support for topological changes
- All components implemented in library form for easy re-use
- Physics model implementation through **equation mimicking**

Object Orientation in C++

Implementing Continuum Models by Equation Mimicking

- Object oriented software design:
  "Recognise main objects from the numerical modelling viewpoint and implement them in code; object = data and functions operating on its data"
- Natural language of continuum mechanics exists: partial differential equations
- Example: turbulence kinetic energy equation

$$\frac{\partial k}{\partial t} + \nabla \bullet (\mathbf{u}k) - \nabla \bullet [(\nu + \nu_t)\nabla k] = \nu_t \left[\frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right]^2 - \frac{\epsilon_o}{k_o} k$$

- Objective: **represent differential equations in their natural language**

```
solve
(
    fvm::ddt(k)
  + fvm::div(phi, k)
  - fvm::laplacian(nu() + nut, k)
== nut*magSqr(symm(fvc::grad(U)))
  - fvm::Sp(epsilon/k, k)
);
```

Object Orientation in C++

OpenFOAM Software Architecture: Layered Development

- Code developed and tested in isolation: natural layered design
  - Vectors, tensors and field algebra
  - Mesh handling, refinement, mesh motion, topological changes
  - Discretisation, boundary conditions
  - Matrices and solver technology
  - Physics by segment: custom (vertically integrated) applications
- Design encourages code re-use: developing shared tools in library form
- Run-time selection mechanism allows new code components without disturbing existing code components: **ultimate user-coding capabilities!**
- Development of physics model libraries with run-time selection
- Mesh handling support: polyhedral cells, automatic mesh motion, topological changes, adaptivity, re-meshing; integrated mesh generation tools
- Custom-written top-level solvers optimised for efficiency and storage
- Substantial existing capability in physical modelling

Software Verification

Software Verification Within Layered Development

- Rule #1: If you have not recompiled or re-linked the code, you have not broken it!
- In layered development, multiple programmers simultaneously work on software components, protected by common interfaces
  - Functionality of components is established as and when they are written
  - Each function throughout the code is consistently marked as `const` access when it is written: this is crucial (retro-fitting `const` is virtually impossible)
  - Unit tests are preserved and automated; testing is performed every time the component or interface is touched
  - Data encapsulation provides compile-level protection: C++ is excellent in following and enforcing data protection rules. No hacking allowed!
  - For multiple run-time model choices, **new code does not disturb existing components**. No recompilation means no new bugs
- If component interface is changed, full verification loop is executed
- Using all possible computer-level verification techniques: `valgrind`, NaN memory initialisation, regular profiling, instrumented class-level debug streams
- However, this is only a basic verification level: complexity brings problems

Software Verification

Software Verification With Generic Programming

- In generic programming, identical lines of source code are used to perform “identical” operations on different types. Code is generated automatically and in a controlled manner by the compiler
- Some examples of generic programming in OpenFOAM
  - `VectorSpace` operations on components, eg. `operator+(scalar, vector, tensor)`
  - Interpolation, eg. upwinding for scalar, vector, tensor
  - Differential operators, eg. `fvm:ddt()`, `fvm:laplacian()`,
  - Boundary conditions, eg. `fixedValue`, `inletOutlet`
- Generic programming significantly facilitates verification at component level: fewer lines of code performing more functions
- Verification is shared among multiple software components and re-used

Conclusion on Verification

- Good software design and “agile programming” renders more reliable code
- **Agile programming**: write and use unit tests before writing source code

Code Validation

Open Source Software Code Validation

- At basic level, validation of Open Source software is identical to other validation efforts
  - Choose well established validation examples
  - Perform detailed CFD studies, with variation of model parameters
  - Perform mesh independence studies
  - Compare CFD results against know reference results and experimental data
- This is a huge amount of effort and cannot be avoided
- Code validation is done in a traditional way
- . . . but is helped by the fact that the top-level code is not monolithic: each physics segment is served by its own top-level solver, eg. `reactingFoam`
- The amount of top-level code is relatively small: all critical functionality is shared and re-used from library components!
- Usually, validation studies are performed in collaboration with end-users or academic partners: reliability of results is well established

Code Validation

Fundamental advantage of Open Source CFD: **Direct Peer Review of Source Code**

- In Open Source CFD, full source code is available for inspection at all levels
- Object orientation and layered design make the code **concise and legible** without reading every line
- Example: `inletOutlet` boundary condition

```
template<class Type>
class inletOutletFvPatchField
:
    public mixedFvPatchField<Type>
{ ... }

template<class Type>
void inletOutletFvPatchField<Type>::updateCoeffs()
{
    const scalarField& phip = this->lookupPatchField(phiName_);

    this->valueFraction() = 1.0 - pos(phip);

    mixedFvPatchField<Type>::updateCoeffs();
}
```

Code Validation

Direct Peer Review of Source Code

- Direct peer review of the source code happens on a daily basis: CFD is still a realm of experts where numerous people understand and care about the details of code implementation
- The code is questioned at every level an on a daily basis (requires effort!) but in terms of code validation cannot be beaten!
- Code validation cases are also available in “full source”
  - Computational mesh (or mesh generation scripts)
  - Material properties, model settings
  - Initial and boundary conditions
  - Discretisation parameters
  - Automatic execution scripts
  - (If possible) Automatic data extraction and comparison with experimental data or reference CFD results
- Validation cases simultaneously serves as a “problem class” tutorial and best practice guideline example: freely distributed and maintained
- Validation cases are maintained by **Special Interest Groups**

Code Validation

Code Validation Contests

- Validation contests are essential and performed regularly with
  - ERCOFTAC Turbomachinery Special Interest Group
  - The Bolund Experiment: Wind on complex terrains
  - Gothenburg 2010, A Workshop on Numerical Ship Hydrodynamics
  - (AIAA CFD Drag Prediction Workshop)
- Provide an impartial forum for evaluating the effectiveness of existing computer codes and modelling techniques
- Allows users to examine the limits of applicability of physical models and CFD without being bound to a single software implementation. Example: bypass turbulence transition modelling vs. natural transition
- . . . but in most cases results are discouraging: typically giving 30-40% influence of **user experience** on results

Naval Hydro Validation

Example: Detailed Validation of the Naval Hydrodynamics Solver

- Validation of the steady resistance free surface VOF solver on a canonical geometry: KRISO Container Ship with known experimental data
- Items under review
  - Mesh refinement study: 6 meshes, 600k to 4.6M cells
  - Variation of Courant number and influence on result
  - Variation of **discretisation settings**: how robust is CFD solution on changes in discretisation settings?
    - Convection and diffusion differencing schemes
    - Gradient calculation algorithm and gradient limiters
    - Boundary conditions
    - Solver convergence criteria
  - Detailed analysis of solution features: far field wave field, stern wetting
  - Stability of solution in time; force decomposition
  - Analysis of simulation time and parallel scaling
- The result of validation study is not only “what accuracy can be achieved” but also “how robust is the CFD result”

Summary and Outlook

Summary on Verification

- Object-oriented software design allows for detailed and critical verification studies, beyond the scope available in procedural programming
- Development and verification of software components is performed from ground up
- Higher programming language software tools built into the compiler help resolve many “old-fashioned” programming problems, eg. data protection
- Code re-use through library components and generic programming exercises the code at higher level
- This can be achieved in any software development framework (not only Open Source)
- . . . but this is only the beginning of Quality and Reliability in CFD

Summary and Outlook

Summary on Validation

- Basic work on code and model validation is always the same
- . . . but the benefit of Open Source is **peer review at source code level**
- The most reliable validation tool is a multi-code “beauty contest” exercise against known validation data, which involves substantial work
- As a benefit, validation cases distributed with OpenFOAM also act as complete tutorial examples and best practice guidelines

Outlook

- In code verification and validation Open Source carries a big advantage: source code is open for peer review at all levels
- Largest uncertainty still comes from the “operator factor”, which remains a prevalent problem for modern CFD