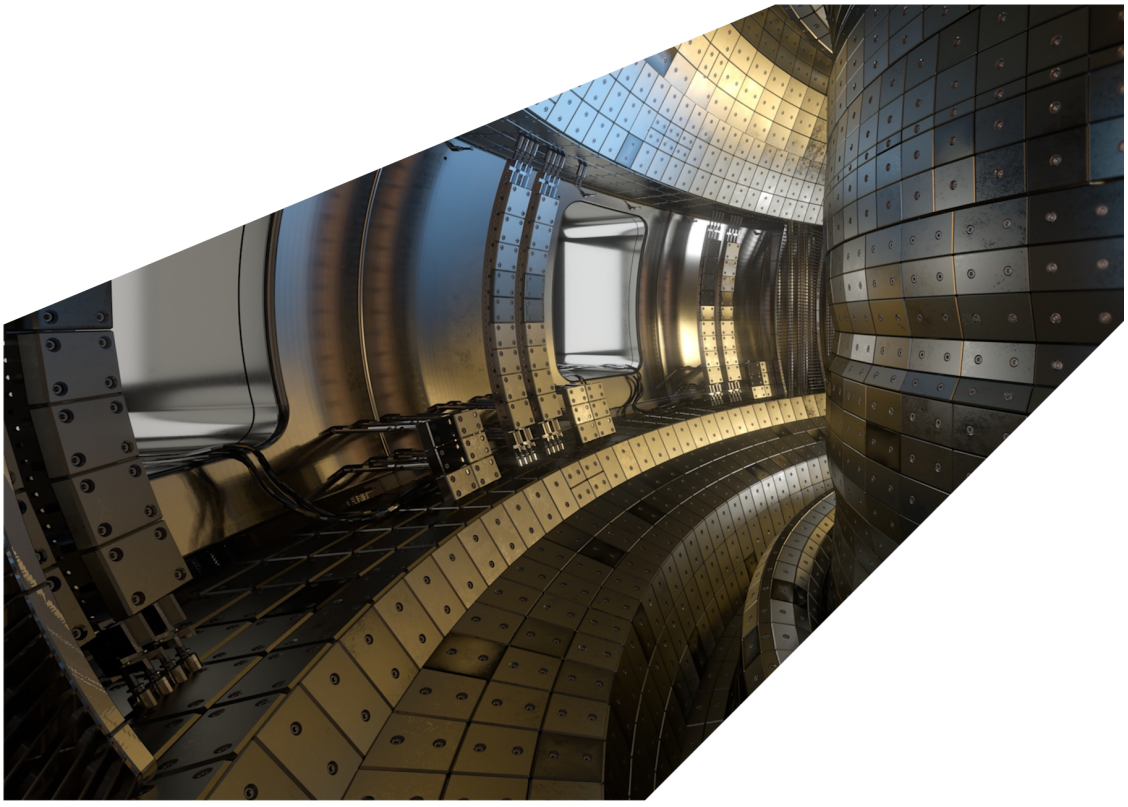
UK Atomic Energy Authority

ExCALIBUR

Report on user frameworks for tokamak multiphysics

M3.1.2

Abstract
The report describes work for ExCALIBUR project NEPTUNE at Milestone 3.1.2. Survey of Multiphysics requirements and available frameworks



UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/EXCALIBUR-FMS/0022
	Issue:	1.00
	Date:	June 30, 2020
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	Wayne Arter Ed Threlfall Joseph Parker Stan Pamela BD	N/A N/A N/A N/A	June 30, 2020 June 30, 2020 June 30, 2020 June 30, 2020
Reviewed By:	Rob Akers Advanced Computing Dept. Manager		August 10, 2020

1 Introduction

The milestone report M3.1.1 [1] highlighted the design challenge represented by project NEPTUNE. The present work investigates the state-of-art in software design for multiphysics applications potentially relevant to the project, focusing on 'software frameworks'. In this document, suitability for deployment at the Exascale is not an over-arching requirement, there is at least equal interest in techniques employed both to ensure a flexible software design, and also those which make the resulting code attractive to others so as to build a wider user and developer community.

Frameworks are defined by Sommerville [2, § 15.2] (after Schmidt et al [3]) as
an integrated set of software artefacts (such as classes, objects and components) that collaborate to provide a reusable architecture for a family of related applications
(In this context, component implies a collection of objects [2, § 2.1.3] whereas class and object are apparently synonymous.)
Sommerville treats frameworks in a chapter on software re-use, consistent with the idea that saving labour is the essence of their attractiveness to others. Sommerville confuses matters slightly by stating first that "frameworks are language-specific", then that a "framework can incorporate other frameworks" in the context where it is clearly indicated that the encompassing framework might be in a different programming language to the others. It does not therefore seem unreasonable to drop the language-specific requirement. It can easily be agreed [2, § 15.2] that "Frameworks are not libraries in that they provide a skeleton architecture for the application", but while normally frameworks will be "implemented . . . in an object-oriented programming language", this does not seem to be strictly necessary.

There will be overlap with M3.3.2 work on design patterns in that the skeleton architecture will usually be a pattern. The pattern however has to be one which dictates the flow of control, and which is extensible. The definition of skeleton architecture extends to demand that the pattern "not be modifiable", whereas it would seem reasonable to allow changes within an overall pattern.

"Multiphysical" software would be better nomenclature than "multiphysics" since there is no restriction to the field of physics in what appears to be a defining bibliography for multiphysics frameworks by Babur et al [4]. In 2015, there were 144 entries in categories listed by ref [4] which included 'Chemistry and Chemical Engineering' and 'Life and Health Sciences', thus fewer than 100 would be relevant to this report. Clearly much of the software listed was likely to have become obsolete even by 2015, although there were a few examples where one framework had morphed into another.

Potentially relevant multiphysics software listed in [4] includes (all C++ unless different language stated in parenthesis after name), with originating organisation before brief description and website:

- PETSc – Argonne (ANL), Portable, Extensible Toolkit for Scientific Computation `https://www.mcs.anl.gov/petsc/`
- FEniCS – (Python, uses PETsc) International, computational mathematical modeling `https://fenicsproject.org/`
- Firedrake – (Python, uses PETsc) Imperial College, solution of partial differential equations

using the finite element method https://www.firedrakeproject.org/

- Trilinos – Sandia, project for parallel solver algorithms and libraries https://github.com/trilinos/Trilinos
- OpenFOAM – ESI group, Computational Fluid Dynamics software https://openfoam.com/
- Nektar++, see Section 2.6 – UK academic, spectral/hp element https://www.nektar.info/
- waLBerla – Germany, widely applicable Lattice Boltzmann from Erlangen https://www.walberla.net/index.html
- LAMMPS – Sandia, Large-scale Atomic/Molecular Massively Parallel Simulator https://lammps.sandia.gov/
- preCICE – Stuttgarts & TUM, Precise Code Interaction Coupling Environment https://www.precice.org/
- POOMA – Los Alamos (LANL), library supporting element-wise, data-parallel, and stencil-based physics computations using one or more processors http://savannah.nongnu.org/projects/freepooma last updated 2005.
- FLASH, see Section 2.3 – Chicago, radiation magnetohydrodynamics (MHD) simulation code for plasma physics and astrophysics, http://flash.uchicago.edu/site/index.shtml
- MOOSE – Idaho, Multiphysics Object Oriented Simulation Environment https://moose.inl.gov/SitePages/Home.aspx
- XMSF – US, Extensible Modeling and Simulation Framework using web services https://sourceforge.net/projects/xmsf/
- OASIS4 – France, exchanges of coupling information between numerical codes representing different components of the climate system https://portal.enes.org/oasis last updated 2019

Selected codes are described in more detail in the sections indicated. Qualifications for inclusion in the list are that source code is available without significant fee, (although it may be necessary to register with the authors prior to download) and that significant updates were made to the code repository in 2020. (The web-sites listed were current as of June 2020.) FLASH is also described in Carver et al [5, § 1] along with Amanzi/ATS, which employs the Arcos framework, see [6] discussed in more detail in Section 2.4.

Babur et al also mention HLA, High Level Architecture, which evolved to become an IEEE standard for distributed simulation, leaving behind its origins as a single piece of software. Although OASIS [7] is present, ESMF, the Earth System Modeling Framework [8] and NEMO, Nucleus for European Modelling of the Ocean NEMO [9], are missing from the bibliography, presumably on the grounds that they are only used by Earth scientists, although both coordinate modelling of many different physical processes. Similarly there are examples of astrophysics codes besides FLASH, such as the Pencil Code [10] which is centred on compressible magnetohydrodynamics and the 'other' NEMO [11] (Not Everybody Must Observe) which is a stellar dynamics toolbox, which are

not mentioned, although they have grown to include a lot of extra physical processes. The paper by Theurich et al [12] indicates that ESMF was important for subsequent developments, and together with the recent survey by Groen et al [13] that covers multiscale computing software, will be discussed further in Section 3.

Regrettably, investigation of the usage of the term "framework" in talks at the SIAM PP20 meeting indicated wide usage as a synonym for library. Here, consideration will only be given to software which represents significantly more complexity than a library plus a test harness. Thus, although there are approximately 50 separate mentions of the term "framework" in the online programme [14], the number of true frameworks in smaller, and the number that are multiphysics is smaller still. The following are noteworthy (author(s) and session(s) at end of line):

- OpenFPM – Dresden, particles and mesh simulation `http://openfpm.mpi-cbg.de/` , last release 2/19 . Incardona, MS22
- Legion – Stanford, data-centric parallel programming system for writing portable high performance programs targeted at distributed heterogeneous architectures `https://legion.stanford.edu/`. Shipman, MS36
- AMReX – ECP, massively parallel, block-structured adaptive mesh refinement `https://amrex-codes.github.io/amrex/`. Gott, MS12, Almgren, MS46

There were other mentions of Adaptive Mesh Refinement (AMR) frameworks at PP20, notably by Shimokawabe in MS25, but most concerned AMReX, which is both the name of the software and the Exascale Computing Project (ECP) co-design centre [15, § 3]. It is notable that the only other mention of framework within the ECP programme [15] is the Multiscale Modeling Framework (MMF) which appears exclusively in the context of E3SM, a cloud-resolving version of CESM, the Community Earth System Model which is compared to the ESMF in ref [12].

The precise criteria for identifying software to be "multiphysics" could be debated. Babur et al include only one current code, OMFIT, relevant to magnetically confined nuclear fusion, consistent with the idea that fusion basically constitutes a single area of physics. Indeed, since NEPTUNE will be concentrating on plasma physics, the use of the term multiphysics in the context of the project might be queried. However, in magnetic fusion work, there seems to be an empirical definition as software which combines the function of two or more existing separate codes. Thus the SOLPS software [16] couples a fluid code (B2) for the ionised component of the plasma with a particle code (EIRENE) for the neutral component. A better definition might be as modelling software capable of treating coupled effects using different physical representations and solution algorithms for the interacting components. Project NEPTUNE software should not become "multiphysics" only when it incorporates plasma chemistry effects.

The following is a representative although far from exhaustive list of frameworks relevant to fusion, again C++ unless stated otherwise in parenthesis:

- OMFIT – (Python) General Atomics (GA), One Modeling Framework for Integrated Tasks `https://omfit.io/`
- BOUT++, see Section 2.5 – York, Plasma fluid finite-difference simulation code in curvilinear coordinate systems `http://boutproject.github.io/`

- ETS – (Mixed) EU, European Transport Simulator within Integrated Tokamak Modelling (ITM). IMAS repository
- JOREK, see Section 2.7 – (Fortran) EU, MHD including more detailed fluid plasma models `https://www.jorek.eu/`
- SMARDDA, see Section 2.2 – (Object-oriented Fortran) UK, Surface interactions of rays and particles. IMAS repository as SMITER kernel
- Alya – Spain, High Performance Computational Mechanics `https://www.bsc.es/research-developmen research-areas/engineering-simulations/alya-high-performance-computational`
- VECMAtk – EU, includes MUSCLE 3, formerly ComPat `https://www.vecma-toolkit.eu/`

BOUT++, JOREK and SMARDDA will be described in more detail in the sections indicated. The IMAS repository is not publicly available, but is usually accessible on request by members of the fusion community to ITER. The Alya software is free subject to a collaboration agreement. Alya [17] and VECMAtk have been included as more general frameworks which have been used for fusion work, respectively reactor modelling [18] and multiscale tokamak core plasma modelling [19]. Neither has yet seen any uptake in fusion outside the collaborators in the EU-funded projects to develop them. The tool-kit VECMAtk is directed towards Uncertainty Quantification and will be described in detail elsewhere.

OMFIT [20] has attracted and continues to attract a very wide range of users. Many of the reasons for this are to be found in the first paragraphs of Section 3.1, but aside from interactivity, one specially highlighted by the authors is that it integrates people engaged in previously fragmented efforts in experimental data analysis, often including multiphysics simulations, into a community based around the software. It goes almost without saying that the larger the usage of software, the more reliable it becomes, provided there is a good management, especially of error reporting. Much of the OMFIT software relates to ancillary tasks such as reading and converting between data formats and visualisation, for which users would otherwise to write their own software, and here OMFIT's comprehensive documentation, extending to YouTube videos, is clearly vital.

The present work contains in Section 2 detailed descriptions of one obsolescent and six other frameworks still under active development. Section 2.8 provides a tabulated overview of the seven. Section 3 summarises what has been learnt about what makes software attractive in Section 3.1, then Section 3.2 indicates how to begin producing complicated designs that nonetheless possess much flexibility, and lastly Section 3.3 sets the scene for the transition to the Exascale.

2 Selected Software in Detail

2.1 OLYMPUS

UKAEA has a history dating back to the 1960s of pioneering software engineering techniques, particularly for nuclear fusion applications. K.V. Roberts promoted what is now known as "literate programming" [21] and subsequently introduced the OLYMPUS programming system [22] which includes design patterns and modules (that could contain more than one subroutine) that constitute a framework within the definition of Sommerville as discussed in Section 1. OLYMPUS is further described in the textbook by Hockney and Eastwood [23, § 3].

The main OLYMPUS design pattern is of enduring interest, especially now that HPC architectures place a premium on managing memory. The assumption is that at heart all physics software has one outermost loop, corresponding either to iterating to a converged solution of the model or to representing system evolution in time, and that convergence or elapse of sufficient physical time may not have occurred when the loop terminates. For efficient use of machine resources, it is desirable that the computation restarts from where it terminated, minimising the amount of information to be stored between calculations. OLYMPUS also allowed for parameters to change at restart, an issue which still might arise nowadays when tracking bifurcations of solutions of nonlinear models.

The design pattern illustrated in Figure 1 handles the restart problem which as indicated has the potential to become tricky, as common routines to read control data (`DATA`) and calculate derived constants (`AUXVAL`) have to be interspersed with others which only need be called at the start of the first calculation (`PRESET`) or at restart (`RESUME`). Thus using the framework eliminates the need for a certain amount of thought, and perhaps more helpfully, a good deal of documentation, since a standard reference can be quoted.

Figure 2 shows a second pattern built around the OLYMPUS library of modules such as `MESAGE` and `HVAR`. The library is of no intrinsic interest now because it is mostly concerned with either timing or string handling. Nonetheless it is important as an early example of 'separation of concerns' to provide portability between machines. In the 1970s the local computing service might well have had to implement the OLYMPUS library in machine-specific assembly code. Nowadays, computer languages have as standard very flexible timing and string handling functions, that emerged out of a set of largely ad hoc libraries like OLYMPUS. Hopefully a similar upgrade path will ultimately be followed by HPC coding of for example array-based manipulations that currently can only be implemented efficiently in an architecture-dependent way.

2.2 SMARDDA Modules

The U(nix)-OLYMPUS tools were developed from OLYMPUS in the late 1980s with the recognition that Unix^{TM} was becoming the default operating system for scientific work. They represented a combination of the OLYMPUS framework above and C shell scripts designed to enhance programmer productivity, both at the individual level, by eg. accepting free format input of FORTRAN code and documentation, and at team level by promoting use of standard data and calling structures, and indeed workflows. The new Fortran standards emerging in the 1990s as Fortran 90 and

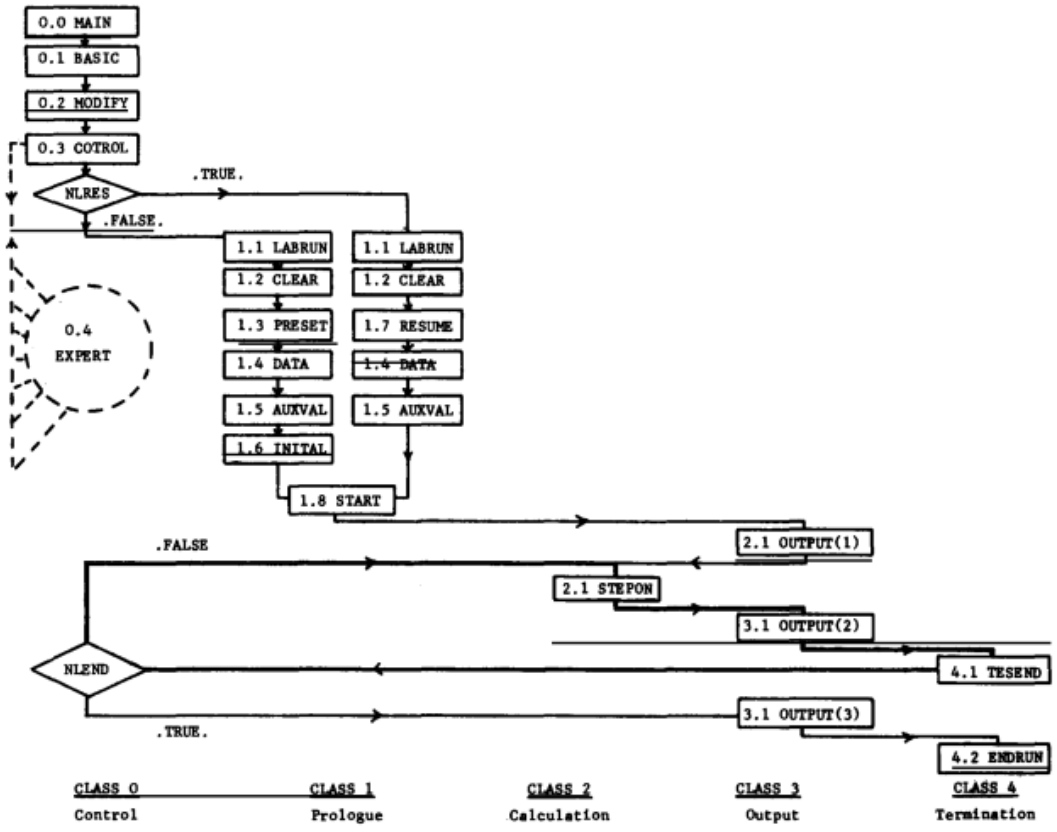


Figure 1: Fig. 1 from ref [22], showing the top-level design pattern. The thick line indicates the main loop.

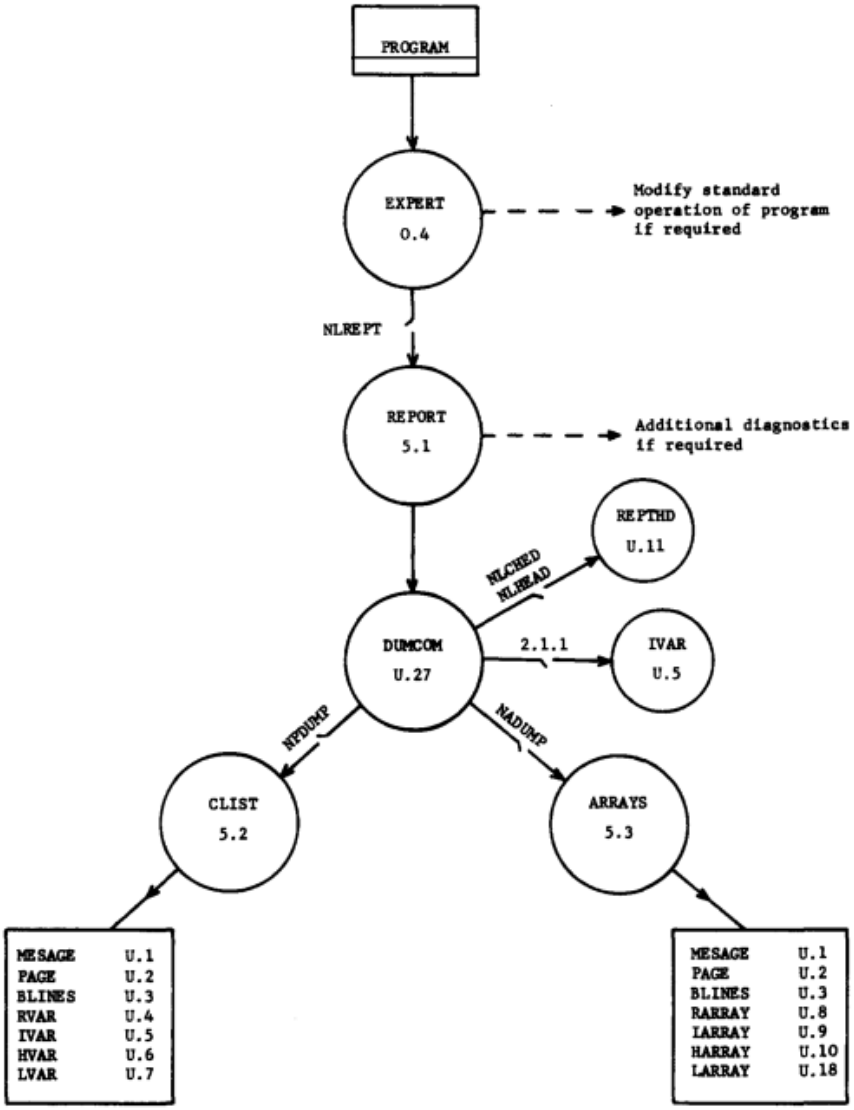


Figure 2: Fig. 4 from ref [22], showing the diagnostic design pattern, and indicating library routines (with the “U” classification).

Fortran 95 reduced the need for many of the U-OLYMPUS features, Linux largely replaced Unix, and U-OLYMPUS was not further updated. For Fortran 95 developments, emphasis switched more on to promoting a coherent style, to exploit the object-oriented features of the Fortran 95 language efficiently. This style was eventually published in ref [24], which includes templates for typical object complexities.

Physics and Scope The SMARDDA modules [25] are a 21st Century development of object-oriented software in this style for magnetic fusion applications. The SMARDDA modules were originally developed for neutronics [26] and neutral beam duct design [27]. As a result of the latter, the basic SMARDDA algorithm was designed to be efficient both for short "rays" representing gyro-orbit segments as well as "long rays" representing neutrals and neutron paths, by combining two pre-existing algorithms, SMART and DDA. Ionised particle trajectories in the ducts were treated using the well-known Boris algorithm [23, $\S$4-7]. In the subsequent development for fieldline tracing [25], the fact that the magnetic field was likely to be a solution of the discretised Grad-Shafranov equation that was only second order accurate in mesh-spacing led to the implementation of a low order Runge-Kutta integration scheme for following the field. An adaptive (Fehlberg or Embedded method RKF23) time-stepping scheme was used purely to avoid the problem of selecting an initial time-step. Hence through a sequence of different applications, software was developed capable of following over 100 million particles on a desk-top in an application to back-scattering electron power deposition in the JET neutral beam ion source [28] and over 2 million fieldlines in an application to JET plasma-facing components (PFCs) [29].

Framework The original OLYMPUS pattern of flow control was not needed by SMARDDA, on the grounds that restarts were unlikely to be necessary, so that the equivalent of the Class 0 and Class 1 are all combined in a single program module that has a $1, 2, 3$ layout, where in sequence order, 1 is initialisation, 2 is compute and 3 is output and closedown. Indeed, strict application of object-oriented principles associates the main loop with a class, eg. the set of all triangles approximating JET PFCs which might receive power along the fieldline through each barycentre. The software was developed as a set of classes, each with its own I/O and constructor/destructor functions as laid out in the templates listed by ref [24], and in one-to-one correspondence with the modules of the software. This classes ranges from the generic, for logging error and warnings, to the generally useful, for representing geometry, to the more fusion-specific such as magnetic field representations, and ultimately the problem-specific, calculating the power deposition in neutral beam ducts or on PFCs. The program module sits at the apex of a hierarchy of classes, meaning that it has to reference ('use') all classes in the hierarchy. It was recently demonstrated [29] that program modules themselves could be easily modified to become classes referred to by a more encompassing program module.

Initially there was a library consisting of routines written in FORTRAN 77, callable by modules written in Fortran 95. In time, it became clear that it would be helpful to construct a library of Fortran 95 modules that could be used by the different applications, see Figure 3. Hence the SMARDDA modules have all the features of a framework, excepting that the skeleton is modifiable within the $1, 2, 3$ layout described above.

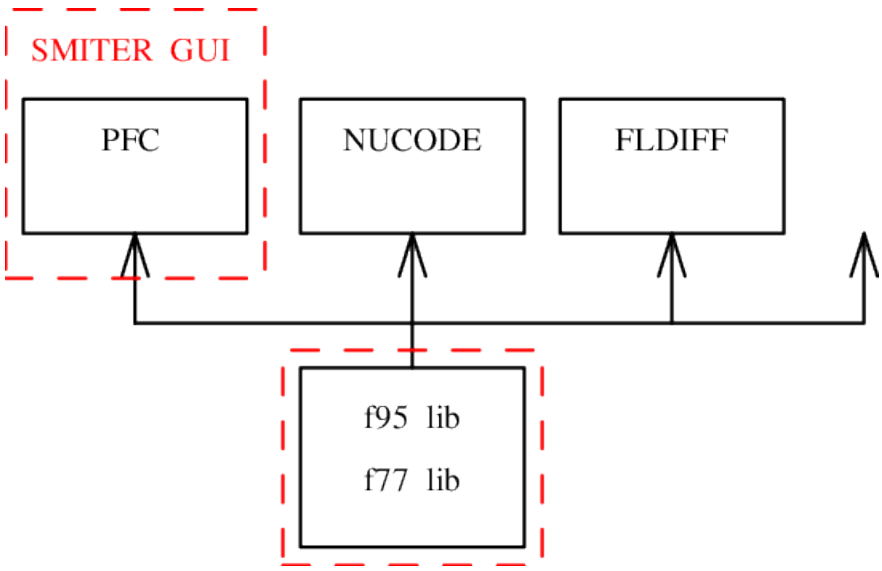


Figure 3: Library structure of the SMARDDA modules. The Fortran 95 (f95) library contains modules used by one or more of application codes SMARDDA-PFC, SMARDDA-NUCODE, etc. (The parts of the software in red boxes form the kernel of the ITER IMAS code SMITER.)

Documentation Regarding documentation, much was produced under contract to ITER intended for designers who would use the software as a 'black box', so-called user documentation. doxygen is used as it fortunately acquired a significant capacity to model Fortran 95 immediately prior to the documentation contract. However, doxygen does not recognise the Fortran namelist feature which is the main way in which control data is input, meaning that namelists have to be crafted into derived types.

2.3 FLASH

FLASH [30] is described:

> *'The FLASH code is a publicly available high performance application code which has evolved into a modular, extensible software system from a collection of unconnected legacy codes.'*

Dubey et al [31] add to the description *'... a Massively-Parallel, Multiphysics Simulation Code'*. FLASH is a community astrophysics code; it has been a clear success in terms of uptake since its inception in Y2K - there are now over 1000 citations on the FLASH website [30].

The code formed when legacy solvers (written in Fortran 77) were refactored into a formal software engineering framework. Subsequent additions to the code have been driven by the physics requirements of the users; Dubey et al [31] references in its title the code's *'Extensible Component-Based Architecture'*.

Physics and Scope Capabilities include compressible hydrodynamics, MHD, nuclear burning, equations of state, radiation, laser drive, fluid-structure interactions, and also particles, which may simply be non-interacting 'tracer' particles. Kinetics (represented by the Boltzmann equation) and gyrokinetics seem absent from the codebase. Gravity (an always-additive and therefore long-range force) is clearly a pressing matter for astrophysics but is irrelevant for NEPTUNE; likewise any cosmolological / relativity physics. The codebase uses only rectilinear computational meshes.

Framework The core algorithms are written in Fortran 90 and wrappers are written in C.

The code is structured into components called Units (many of which implement a specific component of the physics capability; an example is particles) and the code for an individual Unit is demarcated using the Unix directory structure. A build includes only the Units needed for the problem in question. The top level of a Unit's directory defines the API of the Unit. Unit-private data is passed laterally by reference using accessor functions as needed. Subunits allow alternative implementations and also give scope to shared data, avoiding proliferation of accessor functions. Simulations are specified via a collection of config files included at various places in the directory structure: files lower down the hierarchy can override ones inherited from higher up. Users can incorporate novel physics by adding a new Unit.

The framework is object-oriented and encapsulates data; however, this is accomplished using the directory structure and a set of parsed config files (written in a domain-specific language), rather than using an explicit object-oriented language. It is stated in [31] that this may be beneficial for portability.

Each simulation would appear to necessitate compilation of a new executable; the object-oriented character is strictly at compile (i.e. build) time and it seems the code must be rebuilt to change what the simulation does (excepting parameters read in by config files).

There is no GUI; user interface is via text files / command line. Multiple I/O libraries are supported e.g. HDF5 and parallel netCDF (see [31]).

The workflow includes a test suite, which is run nightly. There is an integrated unit test framework and a self-test suite for regression testing. An ensemble of quick-running examples has been proven by a user survey to be a popular feature.

There is rigorous gatekeeping for non-internal extensions to the release version of the code, including a requirement for a specific unit test and a commitment to provide support.

The user community consists of mainly astrophysicists, but there are some other areas of application. There is an email users' group, with support provided by experienced users and the developers. The code is free on request; users may modify but not redistribute their own copies. Documentation includes a User's Guide and also API description extracted using *robodoc*. One interesting aspect has been user surveys, which have provided insight into how the code was used and also indicated the most widely appreciated aspects of the code. [31] cites feedback from close to 300 respondents, which gives the top reasons for FLASH usage as

1. Adaptive mesh refinement (64%)
2. Flexible (47%)

3. Good documentation (47%)
4. Availability of examples (43%)
5. Computationally efficient (37%)
6. Easy to Use (36%)

Summary FLASH is a true multiphysics code and incorporates fields and particles. Its authors claim efficient operation up to tens of thousands of processors ([31] includes an example of 16 million particles running on 32,768 nodes of IBM BG/L machine).

Disadvantages are associated with the rather amphibious nature of the code: it uses build-time methods to achieve object-oriented features from non-object-oriented languages and so must be largely recompiled for each simulation; also, the code is written in more than one language. Writing in Chapter 1 of [5], the authors of the code state that the framework was chosen because the original physics capability was in Fortran and to refactor in an object-oriented language was not an option; it is also conceded that *'... FLASH ... has a big challenge in adapting for future heterogeneous platforms'*. It appears the use of the code may have 'peaked': there are only five publications listed from 2019 and ten from 2018; cf. 93 in 2010 [30].

2.4 Arcos

The US Department of Energy (DOE) develops a suite of software for studying Earth Sciences. Among these are Amanzi [32, 33] and The Advanced Terrestrial Simulator (ATS)[1] [34], a pair of codes which have been developed since 2012. Amanzi solves for flow and reactive transport in porous media, to allow modelling of mineral and contaminant flow through rock and soil. ATS adds the capability to solve effects from ecosystem hydrology, such as thermal processes, evapotranspiration, surface energy balances, and vegetation modelling [34]. Both Amanzi and ATS are written in C++, using modern software engineering standards, and make extensive use of third party libraries such as Trilinos, PETSc and Hypre. Both are open source and available on GitHub [33, 34].

Framework To enable multiphysics simulations with Amanzi and ATS, the Arcos framework [6] was developed to couple the codes. Arcos is a management system with three components, a process tree, a dependency graph, and a state/data manager.

The process tree formally describes the coupling hierarchy of the equation system to be solved. Each leaf node is an equation, while each internal node couples children together to form systems of equations. Each node provides the same interface to its parents, so that equations and systems of equations may be grouped recursively into a hierarchy. This approach merely formalizes the natural structure existing in most multiphysics systems and codes. However, using an explicit, general and dynamically-formed structure means much of the coupling can be automated. It also

[1] "formerly sometimes known as the Arctic Terrestrial Simulator" [34]

allows domain specialists to focus on developing single components without fear of side effects in other parts of the model hierarchy.

The nodes in the process tree only perform administrative work; actual computational work on the equations is delegated to *evaluators*. These evaluators manage one of three kinds of variable, namely 1) independent variables, user-provided functions of the coordinates, 2) primary dependent variables, functions of independent variables for which an equation has to be solved; and 3) secondary dependent variables, functions of other dependent variables. The evaluators are stored in a directed, acyclic graph (DAG) that describes the relationship between variables. Independent and primary dependent variables are leaf nodes in the DAG. All other nodes represent secondary dependent variables, with the directed edges pointing to their dependencies.

While evaluators perform work, they do not store any data (beyond simple metadata). Instead, evaluators access data via the data/state manager, which stores data and controls access to it. This abstracts the physical equations away from the data management, allowing each component to be developed by the relevant expert.

Modularity Arcos' approach is extremely fine-grained. Unlike most scientific software which is modular at the level of equations, Arcos is modular at the level of terms in an equation. This has the following advantages. Firstly, models become more easily interchangeable - for example, pressure depends on temperature and density (in non-isothermal models), or just density (in isothermal models). The evaluator representing pressure will either have or not have an edge in the DAG pointing to temperature, depending on the model. However, other parts of the framework depending on pressure will not need to know whether the model is isothermal. This allows for tight coupling of models, with optimization via lazy evaluation of variables. The dependency graph also means that programmers do not need to manually track and edit code to account for dependencies in different models, reducing bugs and code duplication. This also makes it relatively easy to implement new models at any part of the hierarchy.

The fine-granularity also means that Amanzi/ATS is very amenable to unit testing. Unit testing is difficult in less granular codes, as to test an equation, it must often also be initialized with a mesh, a solver and other components. As the evaluators in Arcos represent a single term, not an equation, and as they hold no data themselves, they are far easier to isolate and test. In addition to a unit testing suite, Amanzi/ATS developers also provide a suite of integrated tests, which double as example problems to aid new users.

Performance portability It is difficult to make statements about performance given the range of models covered by Arcos. However, the Arcos framework has a number of features that are favourable for performance portability. Firstly, by using well-supported solver libraries like Trilinos, Arcos leverages improvements in numerical algorithms with minimal efforts. Secondly, by having a fine-grained heterogeneous structure, Arcos is a good candidate to take advantage of emerging "coarse task" runtime environments. Finally, the nature of evaluators – stateless functors with no side effects – makes Arcos a good candidate for use across a variety of platforms. Evaluators abstract what is done to data from how and where it is done, making it a good framework in which to implement multiple parallelization paradigms (e.g. MPI, OpenMP, CUDA etc.) without intrusion onto the physics code.

Summary The distinctive features of the Arcos framework are its use of a formal structure to describe the equation hierarchy, and its very fine-grained modularity, where every term in an equation is treated as an independent object. This enables a number of features that are desirable in ExCALIBUR. Firstly, the formal graph structure automatically handles dependences in an equation hierarchy, making it easier to implement different models without introducing bugs or code repetition. Secondly, it enables a separation of concerns, with domain specialists able to modify code sections in isolation. Finally, the framework enables performance portability, as the heterogeneous code structure is well-placed to exploit emerging exascale technologies.

2.5 BOUT++

BOUT++ [35, 36, 37] describes itself as

> a framework for writing fluid and plasma simulations in curvilinear geometry. The design is modular, with a variety of numerical methods and time-integration solvers available that can be chosen at runtime. BOUT++ is primarily designed and tested with reduced plasma fluid models in mind, but it can evolve any number of equations, with equations appearing in a readable form. The code is opensource, licensed under the LGPL, and is available from https://github.com/boutproject/BOUT-dev

Framework BOUT++ is a multi-block finite difference / volume PDE solver written in C++, with parallelisation using MPI and/or OpenMP. GPU acceleration is under development. There are optional dependencies on a variety of third-party libraries such as FFTW, SUNDIALS, PETSc, SLEPc. I/O is via netCDF or HDF5.

The core of BOUT++ is a library of functions relevant to plasma simulation in 3-D curvilinear geometry, such as differential operators, definitions of tokamak domains and magnetic geometries (e.g. single/double null), and common boundary conditions. BOUT++ also provides routines for integrating equations in time and for inverting elliptic operators.

As a library, BOUT++ does *not* specify the equations to be solved. These are specified by the user in defining their "PhysicsModel", a class that provides "init" (initialization) and "rhs" (right-hand side) functions. The class is then passed to BOUT++, that is, the user cedes control of the workflow to the library.

In addition to the core library, the BOUT++ project also provides Python utilities for grid generation, post-processing and plotting in tokamak geometries.

Domain-specific language The BOUT++ library routines provide a domain-specific language (DSL), allowing the user to specify equation systems in a human-readable fashion. For example, the wave equation (for amplitude f and velocity g and unit wavespeed) may be written in the user's physics model as

$$\texttt{ddt(f)} = \texttt{Grad\_par(g)};$$
$$\texttt{ddt(g)} = \texttt{Grad\_par(f)};$$

This allows physics models to be implemented in BOUT++ with minimal effort, allowing rapid prototyping. It also means there is a very low bar to entry for new users/non-programmers, and physics models may be implemented with no knowledge of the underlying numerics – for better or worse!

Performance BOUT++ is a versatile framework that may be run across the spectrum of computing system sizes, from laptops to large HPC systems. BOUT++ is currently deployed on Archer (UK Tier-1) and Marconi (Tier-0), among others, and scales to $\mathcal{O}(1000)$ cores for typical problem sizes.

Two of the three dimensions are parallelized with MPI, while all dimensions are parallelized with OpenMP (recasting the 3-D arrays local to an MPI rank as a flat vector). The MPI parallelization in one of the dimensions and the OpenMP parallelization were retrofitted – the anisotropy of plasma in strong magnetic fields meant that in early simulations the resolution used in all-but-one of the dimensions was not sufficiently large to merit parallelization.

Software sustainability and community BOUT++ is freely available via the public Github repository [37], and is licensed under the LGPL. Development is done in public, with regular feedback from community members. BOUT++ uses a development model similar to gitflow, with a stable `master` branch and a development `next` branch. `next` forms the basis for major and minor releases, so new features are introduced into `next`, while bugfixes are introduced into both `master` and `next`. Both these branches are protected, and new code only enters via a pull request, which requires code review and approval from a maintainer before being merged. Travis CI is used to automatically run a comprehensive suite of both unit and system (integrated) tests on every push to the Github repository. Creation and update of pull requests triggers additional tests. Several build configurations are tested, including optimised builds, different compilers and Linux distributions, and linking against various optional third party libraries.

New releases are recorded on Zenodo, allowing each release to have a citable Digital Object Identifier (DOI). Having a DOI for each version is important for reproducibility of scientific results, but producing an accompanying paper for releases simply to obtain a DOI may be an incommensurate amount of work, or not deemed of sufficient interest to be publishable. Reproducibility is further aided by the output of each run recording the git commit hash and the state of the repository ("clean" or "modified").

Documentation is automatically built from doxygen comments in the source code and hosted online at ReadTheDocs [38], while bug reports and community feedback can be done via the Github issue tracker or the BOUT++ Slack workspace.

Hands-on training led by the maintainers is provided for new users at annual workshops, where users can also present their research and discuss new and future code development and features. Research and code development can also be presented at the monthly user meetings, held via video conference.

Summary BOUT++ is a versatile, easy-to-use and reasonably-performative framework. It successfully implements a "separation of concerns" between the roles of user and developer. For users, the DSL allows models to be written in human-readable form, with simple access to complicated geometry-dependent operators. Different numerical methods for discretization and time-integration may be selected at run-time. This means there is a low barrier to entry for prototyping models, and for running simulations on Tier-0/Tier-1 HPC systems. For developers, the source code is freely available on the repository, along with contribution guidelines. There is also an active community of developers around the repository, Slack channel and BOUT++ User Group meetings.

Not all features of BOUT++ are appropriate for ExCALIBUR projects however. BOUT++ emphasizes flexibility rather than performance. One example of this is in the domain specific language where BOUT++ favours a clear interface for the DSL, over optimal performance for any one set of systems. Requiring that operators (such as `Grad_par(g)` above) are independent objects means that the right-hand side functions are concatenations of objects, each being relatively small kernels of work. Joining these into a single loop would require introducing a global index and an outer loop, so operators would be written `Grad_par(g)[i]`. While this could be circumvented with code generation techniques, this has not yet been implemented, and BOUT++ developers favour the cleaner interface to raw performance.

There is also an issue with political control of the users' physics models. The library nature of BOUT++ allows users to develop sophisticated models (e.g. STORM (from CCFE), Hermes and SD1D (from the University of York), plus other models from other individuals/institutions), which themselves require infrastructure like repositories, testing and contribution guidelines. These models have varying degrees of independence from the core BOUT++ framework. Hermes and SD1D are developed by the core BOUT++ developers; STORM is developed in collaboration with BOUT++ developers; in contrast, the source code for physics modules belonging to other institutions are not usually available to BOUT++ core developers. Yet all of these may be presented in papers or at conferences as being "BOUT++". This is a reputational risk to the BOUT++ project. It has been mitigated to some extent by bringing some physics modules into the main repository and testing them, and by endeavouring to collaborate as widely as possible.

2.6 Nektar++

Nektar++ is described ([39]):

> *'Nektar++ is a tensor product based finite element package designed to allow one to construct efficient classical low polynomial order h-type solvers (where h is the size of the finite element) as well as higher p-order piecewise polynomial order solvers.'*

The code is opensource under the MIT licence. It is modern and object-oriented, with the initial release dating from 2006.

Physics and Scope Nektar++ is a spectral / hp element framework for a range of PDEs, which can be hyperbolic, parabolic, or elliptic. The code includes pre-written solvers for acoustic, advection-

diffusion-reaction, cardiac electrophysiology, compressible flow, incompressible Navier-Stokes, linear elasticity, pulse wave, and shallow water problems. The underlying method can be continuous or nodal discontinuous Galerkin. There are the options of implicit, explicit, or IMEX time-stepping (though availability depends on solver choice).

Nektar++ does not currently support particles or any Boltzmann-type equations, although for example the treatment of particle motion in spectral / hp elements is described in the textbook [40].

The solvers can be coupled to provide multiphysics capability. An MPI coupling implementation consists of a smooth interpolating field layer that is receiver-centric in that the receiver is provided with pointwise field data from the sender (then to be resolved to the local basis). Data transfer is handled using the CWIPI library [41, 42]. This framework can couple different Nektar++ solvers, or couple one such to a separate application.

Framework Nektar++ uses a modern object-oriented C++ framework [43]. Multi-threading is via MPI.

The framework is multi-layered in that there is a structured hierarchy of C++ components (much of the structure is evident in the directory structure of the downloaded source code) and the code structure mirrors the mathematical formulation (see library descriptions below). The time-steppers are agnostic to the FEM/solver details and the solvers share much of the underlying FEM codebase. Solvers inherit from base classes, for example one for unsteady flows.

The implementation is partitioned into six libraries, organized into utilities (parallel communications, DFT, maths routines), reference elements, physical elements, domain geometry and mesh, global field data operations and solver base classes.

There is no GUI; input is via XML file (mesh, solver, parameters, boundary conditions); output is via file (HDF5 is supported). Tools are provided for converting input meshes from popular formats (*NekMesh*, which also can make a mesh 'higher order', meaning the inclusion of curved-sided elements) and for converting the output to popular formats (*FieldConvert*). Note that the latest version [44] supports input meshes in HDF5 format in order to avoid the need for a single process to read the entire mesh during setup.

The code is designed to run on anything from a single desktop up to many thousands of processor cores.

The development workflow involves a monthly stable release. There is an extensive testing framework.

The focus of the user community seems primarily engineering and biomedical. The entry barrier is lowered by the availability of a precompiled binary in addition to the source code. Documentation includes a User's Guide and API description extracted using *doxygen*. The code is under active development by groups at the University of Exeter, Imperial College London and the University of Utah.

Summary Nektar++ has a well-designed framework, is written in a clean way in a modern, object-oriented language, and it looks relatively easy to add new solvers. The option to download

a ready-built code (as alternate to the full source) provides an easy entry point for new users for whom there is no need to modify the code. Nektar++ has proven scaling up to tens of thousands of processors.

The caveat is that the existing framework requires new physics to be described by PDEs in up to 1+3 dimensions, ie. maximally time-dependent in 3 space dimensions, and solved using by the restricted set of supported finite-element methods.

There does seem to be some 'price of admission' in terms of user expertise; for example, it is not clear what guidance user gets regarding the maximum timestep in explicit methods to avoid violating the Courant stability limit.

2.7 JOREK

JOREK [45, 46, 47] describes itself as

> The nonlinear extended magnetohydrodynamics (MHD) code JOREK resolves realistic toroidal X-point geometries with a C^1 continuous flux-surface aligned grid including main plasma, scrape-off layer and divertor region. It is based on robust fully implicit numerics, and includes sheath boundary conditions, resistive wall effects, two-fluid effects and neoclassical flows, and particle models.
>
> The well-established physics and numerics community around JOREK has strong connections to the relevant experiments, ITER Organization and the respective ITPA Topical Groups.

Numerics JOREK is an implicit finite-element code written in Fortran, with parallelisation using MPI and OpenMP. It uses a 2-D grid of bi-cubic Bezier elements in the poloidal plane, and Fourier harmonics in the toroidal direction. The fully implicit timestepping scheme leads to a linearised system which is solved by using pre-conditioned GMRES, with a pre-conditioner built by solving independently each individual Fourier harmonic matrix. The code has dependency on a sparse-matrix solver, which at present can be one of MUMPS [48], PASTIX [49] or STRUMPACK [50]. The time-step solver includes the options of Euler, Crank-Nicolson and Gear schemes.

The 2-D poloidal grid is made of quadrangular bi-cubic Bezier elements, which are aligned to the magnetic flux-surfaces, and which can be extended to arbitrary wall surfaces, applicable to any tokamak [51]. A coupled free-boundary module of JOREK-STARWALL [52, 53, 54] is also available to look at resistive-wall effects, and disruptions of a VDE nature (Vertical-Displacement-Event). New geometrical representations are also under development for Stellarators.

The I/O of JOREK is done using HDF5 format. A large number of post-processing tools are also available to look at various aspects of simulation results, like fast-camera imaging, Infra-Red thermography for wall and divertor fluxes, line-profiles, integrals etc.

HPC The JOREK code typically requires an HPC architecture to run advanced cases. Although simple, small test-cases can be run on a laptop, high-resolution requires several thousands of

cores on HPC clusters.

Physics There are a number of physics models addressing a wide range of tokamak applications. The base models are reduced- and full-MHD models. Extended models include

1. two-temperature fluid models
2. two-fluid (diamagnetic) model
3. neutrals fluid model: applied to disruption mitigation and divertor detachment
4. impurity fluid model: applied to disruption mitigation
5. kinetic particles pusher: applied to runaway electrons, impurity transport
6. relativistic electron fluid model: applied to runaway electrons
7. coupled fluid-particles model: applied to TAE's, detachment, ITG turbulence

Development The JOREK code is hosted at ITER, using the integrated platform for code developments. The main features used from the platform are

1. Jira: used to raise, discuss, resolve and track issues from the community, addressing both physics and numerical aspects of the code
2. Bitbucket: used to manage git branches, merges and pull-requests
3. Bamboo: used to schedule automated regression tests for pull-requests

Community The principal coordinator of the JOREK community is Matthias Hoelzl, based in Garching, Germany. The main (initial) author of the code is Guido Huijsmans, based at CEA, France, and at Eindhoven University, the Netherlands. There are a number of code-developer and code-users throughout Europe, Asia and the USA.

Communication is organised via a wiki linked to the website [45] which is restricted to a team of registered users and developers. Meetings of team members typically occur several times per month, to present and discuss various projects and developments. A general meeting is held once per year for one week. There are several mailing lists, eg. one for the entire community, and a helpdesk which includes only expert developers. The restricted JOREK wiki includes a variety of code documentation, ranging from physics equations and other material useful for developers, to user-guides for various aspects of the code etc.

Pros

- High standards of software development
- Well-organised community
- Wide range of applications
- New physics models relatively straightforward to implement

Cons

- Scaling: like all fully-implicit codes, JOREK requires the solving of large sparse matrices, which necessitates large amounts of memory
- At present only one type of finite element is available
- Performance dictates new developments, at the detriment of modularity

2.8 Comparative Table for Different Frameworks

Table 1: Metrics for frameworks described in detail.

Framework	Physics Areas	Language(s)	Docum'n Qualty	Maintenance years approx	User Level(s)	UKRI skills
OLYMPUS	All	FORTRAN	5	50	5	5
SMARDDA	Surface interactions	Fortran 95	4	10	1,2,5	5
FLASH	Astrophysics	Fortran 90, C	5	20	1-5	3 or 4(?)
BOUT++	Tokamak Edge/Scrape-off layer	C++	4	20	2,4,5	5
Nektar++	Fluids FEM	C++	5	14	1-5	5
Arcos	Earth sciences	C++	4	10	2,4,5	1
JOREK	Tokamak Edge	FORTRAN	4	10	4	4,5

1. Documentation Quality
    1 Limited documentation online
    2 Published papers describing use
    3 Extensive documentation online
    4 Published papers describing code in detail
    5 Linked textbook

2. Level of user/developer

    1 Application program level, only changing physics parameters.
    2 Programmer in high level language such as Python.
    3 As 2, but occasional programmer in C, C++ and/or FORTRAN.
    4 Real programmer using mostly C etc.
    5 Developer

3. UKRI skills available

    1 No evidence
    2 Ability to use code
    3 Ability to use code and understand limitations
    4 Demonstrated ability to modify code
    5 Significant part of code written in UK

3 Summary

3.1 Attractive Features

Features highlighted as making software attractive are the presence of a large user-base combined with a strong, well-led development team willing to integrate new code and capable of giving good and rapid support (FLASH, BOUT++, JOREK). First use of code must be easy possibly through containerisation eg. by docker images (Nektar++), but regardless download and installation should be quick and straightforward including a test suite which runs rapidly (FLASH, BOUT++). It should be possible to make significant changes to the physical model by use of a Domain-Specific Language or DSL (BOUT++). Access to free training and to meetings involving other users and developers is also seen as important (FLASH, BOUT++, JOREK). There is an element of chicken-and-egg about this, and indeed all these apart from DSL are not specifically software features. Fundamentally, opensource software has to reward volunteers for doing what commercial vendors charge for, namely maintenance and support, as indicated by the authors of the finite element library with the distinctive name of deal.II in their paper [55] which has findings supporting those of the present report.

At code level, attractiveness would seem to translate into a need for the software to be well-designed so in particular to enable easy addition of new features or incorporation into other codes, and well-documented to make it easy to learn how to use. The history of FLASH is significant for showing the expense, in terms of repeated refactorings each costing up to ten person years of effort, needed to develop a large package from disparate legacy codes. Even so, it seems that codes may have to be specially rebuilt for different applications, and the authors themselves question the cost-effectiveness of a port to Petascale. Equally however it is possible to spend too much time obsessing over an elaborate design which then fails when confronted with a practical application. Project NEPTUNE has adopted a middle approach of a sequence of coordinated core proxyapp developments [56], to avoid this pitfall, consistent with recent thinking on software development that the most effective strategy is some kind of planned 'agile' approach [57, 2].

There is then the big question as to what constitutes well-designed software. If Exascale is not the over-arching requirement, then the use of standard formats and accompanying libraries for input (eg. XML, json) and output (eg. netCDF, HDF5) seems a relatively obvious choice. Further subdivision of the main calculation code into libraries (Nektar++,SMARDDA) would also seem desirable, but the options multiply as this may be achieved, for example by subdivision or layering or a combination of both. As already discussed, a bad first choice may mean a later, very expensive refactoring exercise, even when the code has a more integrated design (SMARDDA). Typically, but not necessarily at a lower level (SMARDDA), there is the need to define different classes or objects, which might also be arranged in hierarchies (layers) and/or grouped (subdivided) in many different ways.

Regarding software design, the book by Hewitt [57] is a very recent work that incorporates this latest thinking, and although some may find that it strays too far into philosophy, it becomes very practical particularly in later chapters. Its main drawback is that it covers all types of software. The earlier material in ref [57] can be very thought-provoking, and in any case it ultimately leads to a requirement to discover who will use the software and what they would want it do, just as demanded by mainstream project management texts, and already initiated for NEPTUNE [58, 59].

A feature not in fact possessed by any of the scientific software studied in detail is a graphical user interface or GUI (although ITER have funded a GUI for SMARDDA). This is possibly because of a need for precision in setting inputs, so that for example sliders and dials are not needed, and also that the setting up of parameter scans can become very tedious if the GUI designer did not anticipate this usage. (This lack is widespread, and the ability to set up scans in a machine-independent way appears to be an attractive feature of VECMAtk provided by the QCG software [60] which comes bundled with VECMAtk.) The modular approach recommended for Fortran 95 design in [24] which allows for separate I/O for each module could be generalised so that instructions for GUI generation were embedded. These would probably mostly take the form of tickboxes for logical variables, and constraints/hints on numerical inputs expected, given selection from relatively small menus of choices for each object.

3.2 Flexibility

As described by Theurich et al [12] in relation to Earth system modelling, the concept of a framework has always been looser than the narrow definition given in Section 1. Thus the ESMF Earth System Modeling Framework is described as principally consisting of a collection of libraries that cover mathematics and computer science as well as physical aspects. Although the survey by Groen et al [13] focuses on multiscale, different physical models often apply on the different scales, hence the paper provides a useful update on some of the packages discussed by Babur et al., and indeed on thinking on multiphysics in general. Again the message is that for physical applications a broader definition of framework is necessary. On the other hand, the strict definition is good for producing actionable software by removing opportunities for developers to make potentially hazardous changes, eg. to the order of execution.

A strong hint as to how achieve flexibility is provided by Hewitt [57, § 7] quoting Bezos' memo to the effect that "Make sure everything you write is a service (API)". This introduces the idea of designing software as a large number of largely independent smaller units which couple together to achieve the final goal, when it becomes important to understand how these smaller units interact. Such interaction can be rigorously modelled and hence controlled using a graph-based approach (Arcos). Indeed graph theory is seen as critical to many aspects of the Exascale, as indicated by the ECP's setting up of co-design Center for Graph and Combinatorial Methods for Enabling Exascale Applications (ExaGraph). OMFIT highlights the importance of a particular graph data structure, namely the tree.

3.3 Exascale

In respect of frameworks, the above has identified not so much the pattern to use at the Exascale, as the key approach to be taken to achieving a flexible but robust design, namely through a graph-based approach like that suggested by Arcos. The aim should be division of the software into relatively small, simple modules that carry minimal information internally. These modules will be arranged hierarchically to form objects and/or grouped to form libraries. Following work on M3.1.3 will seek to flesh out the details and work planned on code generators under NEPTUNE D3.2 will explore the tools available to help achieve good designs.

Inevitably the library aspect of Exascale remains a challenge. It is significant that the authors of FLASH, with experience of multiple historical refactorings, question the desirability of a port of the software to Exascale. Regarding the other current frameworks of more recent inception, there is no indication of any such questioning from their authors, though most if not all seem to have needed updating and refactoring to achieve good performance on present machines operating up to and at the Petascale.

A key challenge of the Exascale is the heterogeneity, namely that different nodes of the computer may have different processors or mixtures of processors and GPGPUs. The relative failure of the Fortran co-array feature [61] indicates the magnitude of the challenge. Co-arrays reserve a privileged array index (appearing separately in square brackets rather than parenthesis) to denote that data is stored on different nodes. This helps developers to "think parallel", but is too simple when it comes to handling a mixed node. It seems progress will come, at least in the short term, not through enhancements to the major scientific programming languages of C++ and Fortran, but through special layers of software to separate user from the machine architecture, to be considered in further detail under NEPTUNE D3.2.

Acknowledgement

The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged.

References

[1] L. Anton, W. Arter, and D. Samaddar. NEPTUNE: Report on system requirements. Technical Report CD/EXCALIBUR-FMS/0014-1.00-M3.1.1, UKAEA, 2020.

[2] I. Sommerville. *Software Engineering. 5th Edition (10th Edition, 2017)*. Addison-Wesley, 1997.

[3] D.C. Schmidt, A. Gokhale, and B. Natarajan. Leveraging application frameworks. *Queue*, 2(5):pp20–75, 2004.

[4] O. Babur, T. Verhoeff, and M.G.J. Van Den Brand. Multiphysics and multiscale software frameworks: An annotated bibliography. Technical Report Computer Science Reports 1501, Technische University Eindhoven, 2015.

[5] J.C. Carver, N.P. Chue Hong, and G.K. Thiruvathukal, editors. *Software Engineering for Science*. Chapman & Hall/CRC Computational Science, 2017.

[6] E.T. Coon, J.D. Moulton, and S.L. Painter. Managing complexity in simulations of land surface and near-surface processes. *Environmental modelling & software*, 78:134–149, 2016.

[7] S. Valcke. The OASIS3 coupler: a European climate modelling community software. *Geoscientific Model Development*, 6(2):373–388, 2013.

[8] C. Hill, C. DeLuca, M. Suarez, A. Da Silva, et al. The architecture of the earth system modeling framework. *Computing in Science & Engineering*, 6(1):18–28, 2004.

[9] Nucleus for European Modelling of the Ocean Consortium. NEMO website. `https://www.nemo-ocean.eu/`, 2020. Accessed: June 2020.

[10] High-order finite-difference code for compressible MHD. Pencil Code website. `https://http://pencil-code.nordita.org/`, 2020. Accessed: June 2020.

[11] Not Everybody Must Observe, stellar dynamics toolbox. NEMO website. `https://github.com/teuben/nemo`, 2020. Accessed: June 2020.

[12] G. Theurich, C. DeLuca, T. Campbell, F. Liu, K. Saint, M. Vertenstein, J. Chen, R. Oehmke, J. Doyle, T. Whitcomb, A. Wallcraft, M. Iredell, T. Black, A. da Silva, T. Clune, R. Ferraro, P. Li, M. Kelley, I. Aleinov, V. Balaji, N. Zadeh, R. Jacob, B. Kirtman, F. Giraldo, D. Mc Carren, S. Sandgathe, S. Peckham, and R. Dunlap IV. The earth system prediction suite: toward a coordinated US modeling capability. *Bulletin of the American Meteorological Society*, 97(7):1229–1247, 2016.

[13] D. Groen, J. Knap, P. Neumann, D. Suleimenova, L. Veen, and K. Leiter. Mastering the scales: a survey on the benefits of multiscale computing software. *Philosophical Transactions of the Royal Society A*, 377(2142):20180147, 2019.

[14] SIAM PP20 Meeting programme and abstracts. `https://www.siam.org/Portals/0/Conferences/PP20/PP20_PROGRAM_COLOR_V4_with_abstracts.pdf?ver=2020-02-06-095558-400`, 2020. Online; accessed June 2020.

[15] F. Alexander, A. Almgren, J. Bell, A. Bhattacharjee, J. Chen, P. Colella, D. Daniel, J. DeSlippe, L. Diachin, E. Draeger, A. Dubey, T. Dunning, T. Evans, I. Foster, M. Francois, T. Germann, M. Gordon, S. Habib, M. Halappanavar, S. Hamilton, W. Hart, Z. (Henry) Huang, A. Hungerford, D. Kasen, P. R. C. Kent, T. Kolev, D. B. Kothe, A. Kronfeld, Y. Luo, P. Mackenzie, D. McCallen, B. Messer, S. Mniszewski, C. Oehmen, A. Perazzo, D. Perez, D. Richards, W. J. Rider, R. Rieben, K. Roche, A. Siegel, M. Sprague, C. Steefel, R. Stevens, M. Syamlal, M. Taylor, J. Turner, J.-Luc Vay, A. F. Voter, T. L. Windus, and K. Yelick. Exascale applications: skin in the game. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190056, 2020.

[16] X. Bonnin, W. Dekeyser, R. Pitts, D. Coster, S. Voskoboynikov, and S. Wiesen. Presentation of the new SOLPS-ITER code package for tokamak plasma edge modelling. *Plasma and Fusion Research*, 11:1403102–1403102, 2016.

[17] M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Arís, D. Mira, H. Calmet, F. Cucchietti, H. Owen, A. Taha, E.D. Burness, J.M. Cela, and M. Valero. Alya: Multiphysics engineering simulation toward exascale. *Journal of computational science*, 14:15–27, 2016.

[18] A. Gutierrez-Milla, M. Mantsinen, M. Avila, G. Houzeaux, C. Riera-Auge, and X. Sáez. New high performance computing software for multiphysics simulations of fusion reactors. *Fusion Engineering and Design*, 136:639–644, 2018.

[19] O.O. Luk, O. Hoenen, A. Bottino, B.D. Scott, and D.P. Coster. Compat framework for multi-scale simulations applied to fusion plasmas. *Computer Physics Communications*, 239:126–133, 2019.

[20] O. Meneghini, S.P. Smith, L.L. Lao, O. Izacard, Q. Ren, J.M. Park, J. Candy, Z. Wang, C.J. Luna, V.A. Izzo, et al. Integrated modeling applications for tokamak experiments with OMFIT. *Nuclear Fusion*, 55(8):083008 (13 pages), 2015.

[21] K.V. Roberts. The publication of scientific Fortran programs. *Computer Physics Communications*, 1(1):1–9, 1969.

[22] J.P. Christiansen and K.V. Roberts. OLYMPUS a standard control and utility package for initial-value FORTRAN programs. *Computer Physics Communications*, 7(5):245–270, 1974.

[23] R.W. Hockney and J.W. Eastwood. *Computer Simulation Using Particles*. IOP Publishing, 1988.

[24] W. Arter, N. Brealey, J.W. Eastwood, and J.G Morgan. Fortran 95 Programming Style. Technical Report CCFE-R(15)34, CCFE, 2015. `http://dx.doi.org/10.13140/RG.2.2.27018.41922`, https://scientific-publications.ukaea.uk/wp-content/uploads/CCFE-R-1534.pdf.

[25] W. Arter, V. Riccardo, and G. Fishpool. A CAD-Based Tool for Calculating Power Deposition on Tokamak Plasma-Facing Components. *IEEE Transactions on Plasma Science*, 42(7):1932–1942, 2014. `http://dx.doi.org/10.1109/TPS.2014.2320904`.

[26] W. Arter and M.J. Loughlin. Radiation transport analyses for IFMIF design by the Attila software using a Monte-Carlo source model. *Fusion Engineering and Design*, 84(1):89–96, 2009. `http://dx.doi.org/10.1016/j.fusengdes.2008.11.051`.

[27] W. Arter, E. Surrey, and D.B. King. The SMARDDA Approach to Ray-Tracing and Particle Tracking. *IEEE Transactions on Plasma Science*, 43(9):3323–3331, 2015. `http://dx.doi.org/10.1109/TPS.2015.2458897`.

[28] I. Turner, R. McAdams, W. Arter, A. Ash, M. Barnard, D. Ćirić, I. Day, D. Keeling, D. King, C. Lane, M. Nicassio, T. Robinson, A. Shepherd and J. Zacks. Ion source backplate loading due to backstreaming electrons and the arc discharge in the JET EP2 neutral beam injectors. *Fusion Engineering and Design*, 148:111273, 2019. `https://doi.org/10.1016/j.fusengdes.2019.111273`.

[29] W. Arter, D. Calleja, H. Leggate, and L. Richiusa. Optimising Fieldline Following Software. Technical Report UKAEA-CCFE-RE(19)04, CCFE, 2019. `https://scientific-publications.ukaea.uk/reports/UKAEA-CCFE-RE(19)04.pdf` to appear.

[30] Anshu Dubey et al. Flash website. `flash.uchicago.edu/site/flashcode`, 2020. Accessed: June 2020.

[31] A. Dubey, K. Antypas, M.K. Ganapathy, L.B. Reid, K. Riley, D. Sheeler, A. Siegel, and K. Weide. Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code. *Parallel Computing*, 35(10-11):512–522, 2009.

[32] J.D. Moulton, S. Molins, J.N. Johnson, E. Coon, K. Lipnikov, M. Day, and E. Barker. Amanzi: An open-source multi-process simulator for environmental applications. In *AGU Fall Meeting Abstracts*, 2014.

[33] Amanzi repository. `https://github.com/amanzi/amanzi`. Accessed June 2020.

[34] ATS repository. `https://github.com/amanzi/ats`. Accessed June 2020.

[35] B.D. Dudson, P.A. Hill, D. Dickinson, J.T. Parker, A. Allen, G. Breyiannia, J. Brown, L. Easy, S. Farley, B. Friedman, E. Grinaker, O. Izacard, I. Joseph, M. Kim, M. Leconte, J. Leddy, M. Liten, C. Ma, J. Madsen, D. Meyerson, P. Naylor, S. Myers, J. Omotani, T. Rhee, J. Sauppe, K. Savage, H. Seto, D. Schwrer, B. Shanahan, M. Thomas, S. Tiwari, M. Umansky, N. Walkden, L. Wang, Z. Wang, P. Xi, T. Xia, X. Xu, H. Zhang, A. Bokshi, H. Muhammed, M. Estarellas, and F. Riva. Bout++ v4.3.1. `https://doi.org/10.5281/zenodo.3727089`, March 2020.

[36] B.D. Dudson. Bout++ website. `https://boutproject.github.io/`, 2020. Accessed: June 2020.

[37] B.D. Dudson. Bout++ repository. `https://github.com/boutproject/BOUT-dev`, 2020. Accessed: June 2020.

[38] B.D. Dudson. Bout++ readthedocs page. `https://bout-dev.readthedocs.io/`, 2020. Accessed: June 2020.

[39] D. Moxey et al. Nektar++ website. `https://www.nektar.info`, 2020. Accessed: June 2020.

[40] G. Karniadakis and S. Sherwin. *Spectral/hp element methods for computational fluid dynamics 2nd Ed.* Oxford University Press, 2005. `https://doi.org/10.1093/acprof:oso/9780198528692.001.0001`.

[41] ONERA. CWIPI Coupling With Interpolation Parallel Interface. `https://w3.onera.fr/cwipi/`, 2020. Accessed: June 2020.

[42] R. Casta and T. Morel. Test de la fonctionnalité ordre élevé du coupleur de codes CWIPI et Intégration dans OpenPALM. Technical Report TR-CMGC-19-95, CERFACS, 2019.

[43] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, and S.J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205–219, 2015. `https://doi.org/10.1016/j.cpc.2015.02.008`.

[44] D. Moxey, C.D. Cantwell, Y. Bao, A. Cassinelli, G. Castiglioni, S. Chun, E. Juda, E. Kazemi, K. Lackhove, J. Marcon, et al. Nektar++: enhancing the capability and application of high-fidelity spectral/hp element methods. *Computer Physics Communications*, 249:107110, 2020.

[45] Jorek website. `https://www.jorek.eu/`, 2020.

[46] G.T.A. Huysmans and O. Czarny. MHD stability in X-point geometry: simulation of ELMs. *Nuclear Fusion 47, 659*, 2007.

[47] O. Czarny and G.T.A. Huysmans. Bezier surfaces and finite elements for MHD simulations. *J. Computational Phys. 227, 7423*, 2008.

[48] Mumps website. `http://mumps.enseeiht.fr/`, 2020.

[49] Pastix website. `https://gitlab.inria.fr/solverstack/pastix`, 2020.

[50] Strumpack website. `https://github.com/pghysels/STRUMPACK/`, 2020.

[51] S. Pamela, G. Huijsmans, A.J. Thornton, A. Kirk, S.F. Smith, M. Hoelzl, and T. Eich. A wall-aligned grid generator for non-linear simulations of MHD instabilities in tokamak plasmas. *Comp. Phys. Comm. 243, 41-50*, 2019.

[52] M. Hoelzl, P. Merkel, G.T.A. Huysmans, E. Nardon, E. Strumberger, R. Mc Adams, I. Chapman, S. Gunter, and K. Lackner. Coupling the JOREK and STARWALL Codes for Non-linear Resistive-wall Simulations. *Journal of Physics: Conference Series 401, 012010*, 2012.

[53] F.J. Artola, G.T.A. Huijsmans, M. Hoelzl, P. Beyer, A. Loarte, and Y. Gribov. Non-linear magnetohydrodynamic simulations of Edge Localised Modes triggering via vertical oscillations. *Nucl. Fusion 58, 096018*, 2018.

[54] F.J. Artola, K.J. Lackner, G.T.A. Huijsmans, M. Hoelzl, E. Nardon, and A. Loarte. Understanding the reduction of the edge safety factor during hot VDEs and fast edge cooling events. *Physics of Plasmas 27, 032501*, 2020.

[55] W. Bangerth and T. Heister. What makes computational open source software libraries successful? *Computational Science & Discovery*, 6(1):015010 (18 pages), 2013.

[56] W. Arter, L. Anton, D. Samaddar, and R. Akers. ExCALIBUR Fusion Modelling System Science Plan. Technical Report CD/EXCALIBUR-FMS/0001, UKAEA, 2019.

[57] E. Hewitt. *Semantic Software Design: A New Theory and Practical Guide for Modern Architects*. O'Reilly Media, 2019.

[58] D. Samaddar. NEPTUNE: Report on Y1 2019 Internal Workshop. Technical Report CD/EXCALIBUR-FMS/0018-M1.1.1a, UKAEA, 2020.

[59] D. Samaddar. NEPTUNE: Report on Y1 2020 External Workshop (REPORT1). Technical Report CD/EXCALIBUR-FMS/0010-M1.1.1b, UKAEA, 2020.

[60] Poznan Supercomputing and Networking Center. QCG website. `http://www.qoscosgrid.org/trac/qcg`, 2020. Accessed: June 2020.

[61] R.W. Numrich. *Parallel Programming with Co-arrays*. Chapman and Hall/CRC, Boca Raton, 2019.