Challenges in Large Scale Simulation Software Projects. An Exemplary Analysis of the CFD Toolbox OpenFOAM

Author:
Caspar Kielwein
Matr. Nr.: 280762

Supervisor:
Prof. Dr. Uwe Naumann

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

Table of Contents

1. Introduction:

“Many scientists and engineers spend much of their lives writing, debugging, and maintaining software, but only a handful have ever been taught how to do this effectively: after a couple of introductory courses, they are left to rediscover (or reinvent) the rest of programming on their own. The result? Most spend far too much time wrestling with software, instead of doing research, but have no idea how reliable or efficient their programs are.” —Greg Wilson [Wilson, 2006]

A lot of necessary effort in simulation science and numeric is in fact involved in developing software, more than in developing physical models mathematical methods. Many a man-hour in the scientific community concerned with simulations is spent in designing, coding, fixing and maintaining software. With the rising effort needed to develop and maintain scientific software it is necessary to do so in a systematic and quantifiable way i.e. scientifically. The application of such an approach is combined in the term Software Engineering, the application of the engineering approach to the development of software. Software engineering describes the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software, and the study of these approaches. The term software engineering was coined in the 1968 NATO Software Engineering Conference [NATO, 68]. Software engineering has been developed address the fact that increasingly large scale software induces large scale and complex development processes, which require a systematic approach to insure high quality of the resulting software. Over the years, the discipline of software engineering has created a number of rules and best practices, which help to tackle the challenges of developing and engineering software.

Since its creation software engineering has become a widespread mean to enhance the development process as well as the quality of the resulting software. It offers powerful tools to enhance the development of any software. Scientific software and simulation software in particular need additional attention as their development poses additional challenges, which are not as much present in common industrial software development processes. These challenges come among others from the fact that scientific software has some inherent experimental nature, as it is on forefront of scientific discovery by its nature of being scientific software.

The usefulness of software engineering has been accepted in software industry and the according methods are in prevalent use. Over the time it has shown, that the application of software engineering techniques to software development projects deliver advantages in the cost of the design as well as the quality of the results, when compared to a less systematic more ad-hoc approach. Most large scale software projects are now embedded in particular design processes and subject do several metrics and other considerations of software engineering. The scientific sub-disciplines of computer science, software construction and software engineering have developed many means to enhance the development process of software. These include programming languages, coding practices, designs of software architectures, the design process itself as well as a multitude of tools.

Since there is no technique of software engineering which can be applied well to tackle all problems (often called 'silver bullet'), these techniques have to be adapted to specific domains and kinds of software. Many such techniques have been developed, with a certain application in mind. There are for example special architectures like Enterprise Java Beans [EJB, 2013] for database applications as well as design processes like the Unified Software Process for the development of these applications. For any new field of application software development requires new techniques of engineering, which fit well to this particular application.

In the field of scientific software the use of software engineering techniques still has not yet spread that far. Simulation Software has fundamental differences to conventional software in all layers of its development. The requirements for the actual code, the software architecture as well as the design process have aspects specific to scientific software. This paper describes the current state of the art for the development of scientific software, the particular challenges which have to be faced and possible new directions for software engineering in scientific software.

2. Challenges in large scale simulation software

Scientific software projects can be distinguished by characteristics that clearly separate them from other development projects like open source or commercial software development processes.

2.1. Student participation

Like all software projects, scientific software projects are very diverse. They vary in size, lifespan, application domain and used base technology. Therefore it is impossible to define one single fixed development process that fits for all those projects. Anyhow, scientific software projects are distinguished from industrial software projects by the fact that they are carried out with the participation of students. Main parts of the functionality of many scientific software projects are developed in students' theses or with the help of student developers. This has two main consequences [Hoffmann, Lichter, 2011]:

1. Students typically join the development team just for their thesis and are seldom a part of the development team for more than nine months. Thus students often only contribute to a very limited part of the project, namely the specific feature that is being developed in the context of their thesis. This means that students often lack a broad overview over the project and have little interest in the overall project beyond the part they are working on.
2. Second, students are usually not experienced. Many students have not been in any real world software development project before their thesis. They often lack software engineering know-how and have little to no experience with tools, languages, libraries or frameworks that are used in the project.

Scientific software projects are comparable with open source software projects as in these the number of collaborators is also higher then in common industrial software projects. This indicates that development processes for scientific software projects could similar to those used by open source software development projects.

2.2. Embedded in Science Projects

Scientific software projects are usually embedded in one or more research projects. [Hoffmann, Lichter, 2011] This impacts the software and the underlining development process, since the direction of research my change based on obtained results, new research opportunities or funding. These changes in the research project typically shift the focus of the software and often induce changes in requirements.

Additionally not every research produces the expected results. Empirical studies may uncover misconceptions in earlier research and may deem old concepts and ideas infeasible. Fatal flaws in numeric schemes might be uncovered etc. . The use of simulation software is also often first of a kind and can thus encounter problems, when computational boundaries are encountered. Scientific software projects and simulation projects in particular have thus to deal with dead-ends and throwbacks regularly.

Parallelism

Simulation software is often designed for machines with multiple cores which go up to massive parallelism in the case of supercomputing clusters. Since code for parallel machines is both, more complex to implement as well as more difficult to debug and maintain, simulation software, which has to implement parallel functionality incorporates additional technical difficulties.

3. Challenges in Management of Scientific Software Projects

Any development process aiming to support and guide scientific software projects has to consider these special characteristics These characteristics translate into the following risks and challenges for the management of scientific software projects. In project management so called risks are assessed for the project, which incorporate the expectations and possibilities for all that could go wrong in the project. Risks come from different sources, some stem from human factors, other are technical, some come from internal sources of the project, others from the outside. Risks need to be addressed according to the probability of their occurrence as well as their severity. For every risks appropriate mitigation strategies should be formulated in the management of the project.

3.1. Higher Technical Risks

Since scientific software projects are often of experimental nature, the prevalence of technical risks is higher than in conventional software development. In particular when new scientific and/or mathematical methods are implemented one cannot guarantee the feasibility of the problem until nearly complete development of the software. In the case of simulation software the need for parallel computing enlarges the technical difficulties of the software development.

This means that project management for scientific software projects has to prepare for frequent setbacks and necessary changes in the software's functionality, when older concepts show to be infeasible. The software's architecture has to allow easy extensions and changes in the code base to accommodate for this. Also the development process of the software project should be designed in such a way that setbacks due to the occurrence of technical risks do not throw the whole project back more then necessary. In general there should be constant awareness for these risks and possible mitigation strategies. New developments in the scientific fields related to the software project should be accounted for quickly and communicated throughout the development team.

3.2. Higher Person Risks

The fact that scientific software projects are often realized with a strong student participation introduces additional risks, which stem from the inexperience with software engineering in general and the field of application of the specific project of most students. The quality of the code and other artifacts delivered by students cannot always be held to the same standard as those by more experienced coworkers. This is similar to open source projects, where the qualification of collaborators cannot be enforced beforehand. The development process of scientific software projects thus has to incorporate additional measures to control and validate the quality of the delivered artifacts.

3.3. Changing Requirements

The fact that scientific software projects are embedded in general scientific projects makes them much more subjected to changing requirements then industrial software projects. Funding is usually not as project specific as in industry. New scientific developments might introduce new requirements even late in the development or make features of the software redundant, when underlying understanding of the problems at hand changes. This either eliminates old requirements or introduces new ones to incorporate new findings.

Scientific software projects are thus even more subjected to changing requirements then most industrial software projects. These changes may occur throughout the whole design process and life-cycle of the software. Software engineering techniques for scientific software thus have to be very flexible and be able to quickly adapt to changing requirements.

Acknowledging these challenges and risks leads to a set of non-functional requirements for the code, architecture and design process of large scale simulation software, which is to be developed:

The **Code** of the software has to be efficient and in many cases support parallel computation. Also the code should be especially easily understandable allow student workers to be effective developers.

The **Architecture** has on one side to support efficient algorithms and implementations. On the other side it has to be easily extensible to accommodate for new functional requirements and flexible to allow easy changes, when necessary. At last it has to be easily understandable to allow the frequently changing development teams of scientific software to work effectively.

Design Processes for large scale scientific/simulation software projects have to enforce these requirements. On the other side the design process has to accommodate for high technical risks and supply necessary mitigation strategies. It also has to incorporate the introduction of new students to the project and manage the fluctuations in the development team. The design process again needs to be highly flexible.

The next chapter explores how these requirements have been tackled in present literature.

4. Current State of the Art

While there is an abundance of scientific as well as more casual literature on design processes and software engineering in general, the literature for the development of scientific software is sparse at best. Literature concerned with the development of simulation software in particular very rare go beyond describing algorithms and coding techniques. There are several publications, which state the need for systematic software engineering in the domain of scientific software, but there is currently no comprehensive work, which collects and describes such techniques.

A search for scientific literature on architectures for scientific software yielded no results, which concern a general approach and only very few and basic applications. Among these is *OpenFOAM: A C++ Library for Complex Physics Simulations* [Jasak, 2007] by Jasak, Jemcov and Tukovic, which describes some of the rationales behind design decisions for the CFD library OpenFOAM.

In the case of design processes for scientific software the situation is even more disappointing. Among the very little literature found is *Processes and Practices for Quality Scientific Software Projects* [Hoffmann, Lichter 2011] by Hoffman and Lichter.

There have also been some developments from the domain of computational biology among which are *Where's the Real Bottleneck in Scientific Computing?* [Wilson, 2006] by Gregory Wilson and *Scientific Software Development Is Not an Oxymoron* [Baxter, 2006] by Baxter et al. ,but again these works seldom go beyond stating the need for specialized software engineering techniques for scientific software.

This chapter will shortly present the current state of the art in software engineering for scientific and simulation software. It roughly divides it into the code, the architecture and the design process. A design process model proposed by Hoffman and Lichter is presented.

4.1. Simulation Code:

Many of the rules for writing efficient programs also apply to the implementation of simulation software. There are some problems, which occur particularly often in simulation code. This work will concentrate on these and not on efficient code in general. The computational bottle-necks of simulation software usually lie in the manipulation of large matrices and solving large equation system. On an implementation level, this involves manipulation of large often multidimensional arrays of floating point numbers.

The specialties of numeric code have been known for some time now, since mathematical programs have been in development and use since the beginning of computers. There are languages like FORTAN, which have been designed with the usage in numeric software in mind as well as more specialized simulation languages like MATLAB. Of all the challenges in simulation software projects, those of writing good numeric code are those, which are known best. The need for efficient code is rather obvious and the drawbacks of inefficient code are easily recognized: slow run-time up to the point of making the program useless. Since the need for efficient code has been present since the beginning of the use of computers, there is a lot of general knowledge and literature about the topic. The different techniques to write efficient code, like good cache use are also easily understandable from a technical standpoint albeit not that simple to apply correctly. Many of these techniques to increase the codes efficiency are specific for the programming language used and can thus not be given as general information. Among the most important guidelines for simulation code which hold for pretty much all programming languages are the following:

Efficient Cache Use:

The time needed for transfer data to the CPU from more distant storage often exceeds the time needed for the actual computation. It is thus necessary to store the data, which is used often in memories with fast access times. Modern computers have several layers of memory with different sizes and access times. Values are loaded into the higher level caches by specific cache paging algorithms and the efficiency of the code can be greatly increased if mostly variables from these caches are used for computation and these caches are refreshed as rarely as possible. The cache paging algorithms usually load more then one value into the cache.

Variables should be stored in memory in such a way that those, which are used in computation together are closely are loaded into the cache together. This means for example that multidimensional arrays should be traversed in the order in which the values are used. In C and C++ this means that the last indexes should be used for inner loops while for FORTRAN it is the first index [Oliveira, 2006]. This is due to the way these languages manage matrix indexing.

Avoid Overheads:

One source of overheads comes from function calls. It is thus useful to try minimizing the number of function calls although this has to be done with care, not to lower the overall quality of the code. This overhead can be eliminated by in-lining the function. This means, that the code of the function itself is inserted in every place, where the function is called. This can either be done by the programmer directly, or by the compiler.

Another kind of overhead is introduced when object oriented code is used. The overhead is in object creation and destruction. To keep the code efficient, these operations should be minimized. To achieve this, the programmer has to take care, when using virtual functions and allowing several layers of function overloading, as this increases the amount of overhead.

4.2. Architecture:

While guidelines for the code of simulation software have been explored in depth, there has been much less research in the field of architectures of simulation software. As of state January 2013, there are no publications which deal specifically with the design of architectures for scientific software or with the design of architectures for simulation software in particular [Hoffmann, Lichter, 2011]. Still some methods and best practices can be found in publications with subjects on related terms.

The key concept to the design of architectures for simulation software is object orientation [Jasak, 2007]. While object oriented design of architectures as a concept is known to many developers of simulation software, the principle is often not followed through consistently.

Object oriented design has clear advantages as it enables the creation of more flexible programs, which are easier to maintain and extend. It also allows a close mapping from the domain of the software (equations, numeric methods in the case of simulation software) to the architecture. This again makes the code easier to understand as well as to change and extend, as the interfaces between the classes closely resemble the relationship between the underlying concepts. Encapsulation hides all inner data and functionality from the user, whose knowledge of the general behavior of the solver is sufficient to apply it. This is often not the case in procedural designs.

The benefits of object oriented design and programming have been largely accepted in the software industry. The concept has also been successfully extended to that of “component based software” ,which is seeing prominent use in enterprise systems. Most scientific software is not (yet) developed using object oriented design and programming [Oliveira, 2006]. One reason for this are the perceived drawbacks in efficiency of object oriented code. Since object oriented code introduces new computational overheads in object construction and destruction the need for efficient code requires that these overheads are minimized. This means, that the architecture should be designed in such a way, that not a large amount of objects are created during run-time. This goes beyond the need for efficient code, as the amount of objects is largely determined in the design phase of the software process before the according classes are coded themselves.

4.3. Design Process:

Design processes and process models are the least explored part of scientific software projects. Many scientific software projects don't define a dedicated development process or implement a process that doesn't cope with the specific constraints of scientific software development. The typical course of designing a scientific project makes it hard to directly apply those models which are well known in software industry.

Hoffmann and Lichter [Hoffmann, Lichter, 2011] propose a design process model for developing scientific software in the context of general scientific projects. The process was designed to explicitly incorporate the fact that these software projects are contained in scientific projects and thus undergo continuous changes and are executed with a large student participation. The authors recognize that scientific software projects resemble open source projects in many ways. They are evolutionary, open, focused on an extensible infrastructure to react on usage shifts, suffer from quick changes in the development team and mainly develop functionality in dedicated sub-projects.

They propose a framework, which divides the complete software development process into three sub-processes:

- **The feature process(es):** For every new feature of the software a new feature process is initialized and maintained. The process is managed by the feature team, the group of people working on designing and implementing this particular feature. These feature teams often contain few permanent members of the development team and a higher number of students.
- **The platform process:** The objective of the platform process is to ensure that a reusable platform (called the platform feature) is continuously developed and maintained. The platform process is started ad the beginning of the scientific software project and maintained until the project ends.
- And finally the **Coordination and Collaboration process** manages and coordinates the features, which evolve during the development of the software. It defines all process rules, documentation standards and templates for the platform, as well as the feature processes themselves. Furthermore it establishes and manages the development infrastructure. Thus it incorporates configuration and change management as well as deployment infrastructure and development architecture. Furthermore this process determines the projects time planing, i.e., synchronization points and milestones with the according deadlines for the platform and feature projects are managed in it. Finally it defines global management tasks for the coordination of development teams. The coordination and collaboration process is typically organized by a coordination team which is created out of senior developers from the feature and platform teams.

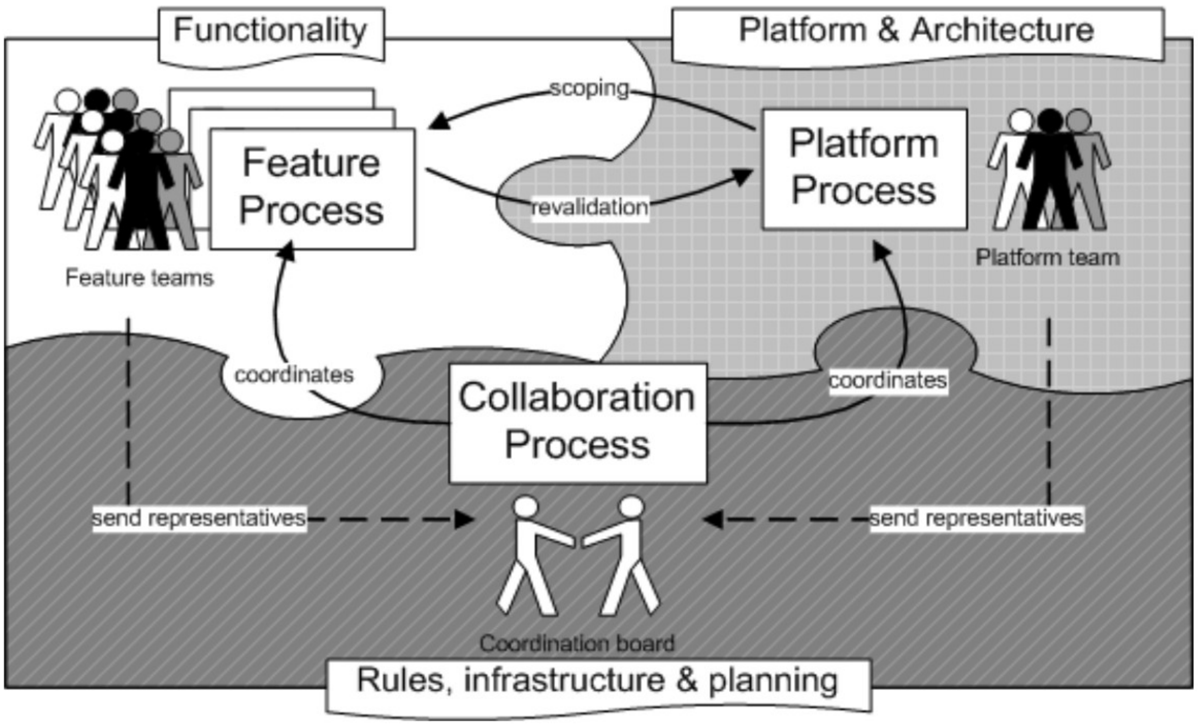

Figure 1: Dependencies between sub-processes [Hoffmann, Lichter, 2011]

Suggested Best Practices

Hoffmann and Lichter propose a set of best practices to support the design process. The best practices Baxter et al. [Baxter, 2006] propose are mostly contained in this more elaborate set. Although the later do not propose a complete design process model but only this set of best practices. These best practices serve to make the development process flexible enough to be able to face the challenges of scientific software development, while being rigorous in enforcing the necessary rules to sustain middle to large scale development projects.

Sprint based release management: Industrial strength release planning is typically inadequate for scientic software projects where features can quickly emerge and disappear again because it is too heavy-weight and inflexible. The authors therefore recommend a lightweight release planning similar to the agile concept of Scrum [Scr]. The suggested length of each sprint is up to three month since, the development speed in scientific software projects is often slower then in industrial projects.

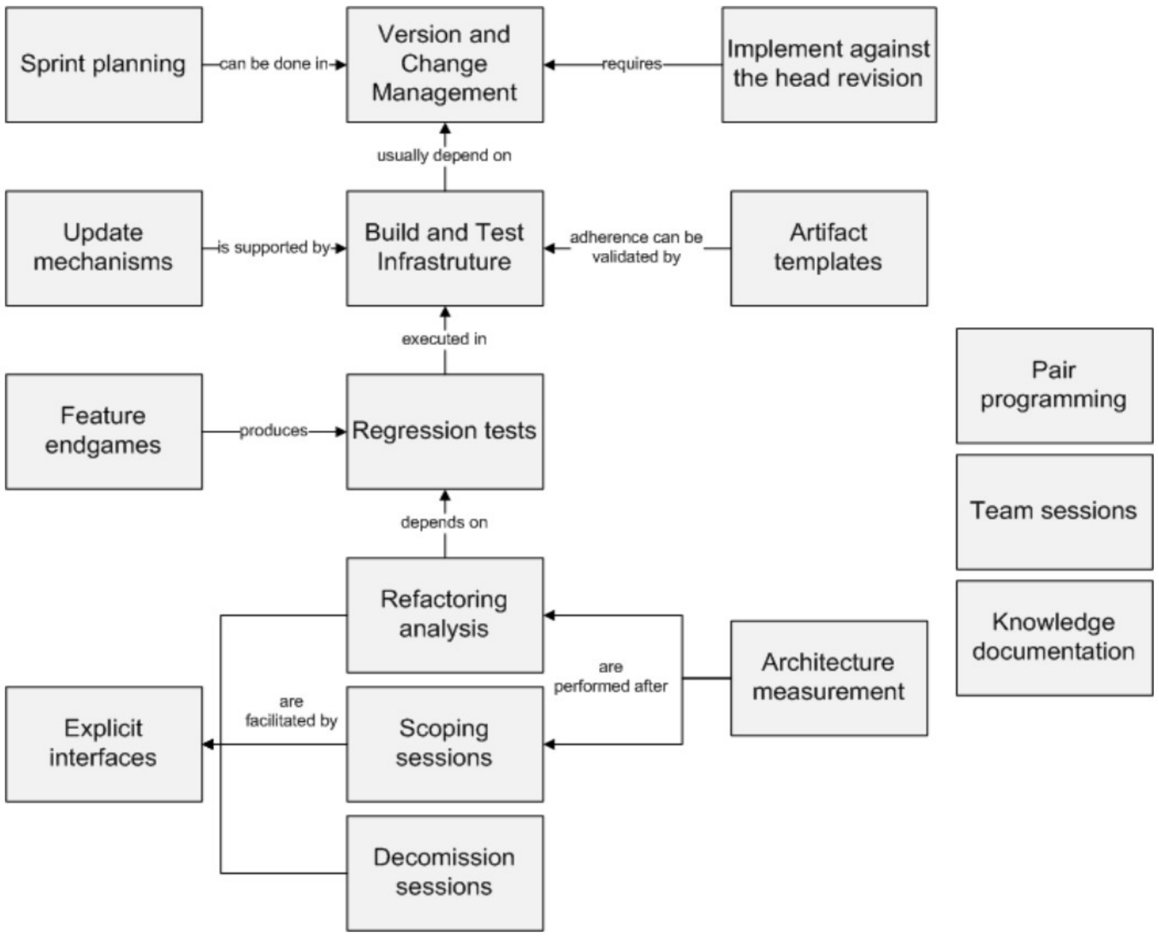

Figure 2: Best Practices in Context [Hoffmann, Lichter, 2011]

Integrated change management: Since scientific projects are often performed in a decentralized and evolutionary way, the management of changes and their impact to existing project artifacts is crucial. This demands a close integration of version and change management systems. Every bug or enhancement request should be handled by a ticket of the change management system and every change to a project artifact must be associated to a respective ticket. Moreover all artifacts (including source code, documentation, test cases and examples) should be under version control. This ensures that non-code artifacts, like definitions and documentations are kept up to date. It also enables the old and new developers of the project to have access to all information and enables easier introduction into the project.

Use automated build and test systems: The authors suggest that an automated build system should be used to assure the integration of all features including those that are under development. The build system should perform nightly builds to assure compilability and integration of components and should create release builds on a regular basis alongside each sprint. Furthermore they recommend adding automated regression tests and static analysis performed after each successful build.

Provide update infrastructure: Since scientific software changes and evolves quickly an update infrastructure should be provided once the software is released. This ensures that the software follows the scientific progress during the whole life-cycle for all users.

Perform quality assurance phase for each feature project: Every feature project should plan a feature freeze and a quality assurance phase at the end of its development. This phase assures three important quality aspects. First it checks that required functionality is implemented and the feature is integrated in the platform correctly. Second it assures that the architectural of the feature adheres to architectural rules of the platform process. At last it makes sure that sufficient documentation is available. Since feature projects are often performed in the context of students' thesis works and students are usually unavailable after the end of the thesis a strict quality assurance phase is vital for the overall project success. Otherwise any information that is not handed over is lost and could be recovered only with substantial effort.

Measure architectural quality: To maintain a sustainable architectural in the dynamic and changing environment of scientific software projects, the conformance of the current code basis to the architectural definition must be measured and checked regularly. There are several tools available to measure and asses software architectures e.g. [Hello2Morrow, 2010] [Metrics Project, 2010].

Explicitly define interfaces of features: The interface for every feature should be defined explicitly before the implementation of this feature. All access to that feature should only be allowed through those interfaces. This ensures that the impact of changes of a feature to other features is small and manageable.

Define templates and rules for artifacts: All kinds of project artifacts, code and non-code, should be developed according to present templates and respective sets of rules. Templates increase the readability of documents, greatly ease identification of changes in version management and simplify introduction into the project for new collaborators.

Perform regular platform scoping: The platform contains reusable code that should be used by all feature projects. Its main purpose is to prevent redundant developments in different feature projects. Therefore special platform scoping sessions should be performed on a regular basis to transfer reusable parts from the feature processes to the platform. Closely related to the scoping sessions are the next to practices:

Check for obsolete features: Since resources in scientific software projects are often short, no effort should be wasted to develop or maintain features that are no longer needed. Features may become obsolete either because the usage of the tool has changed for example when requirements to the software change or my be replaced by a new feature or third party product.

Perform refactoring analysis's: Every time a new feature is to be implemented the platform team should ensure that the platform architecture is adequate to the implementation of the new feature. If this is not the case refactoring steps should be identified to improve the architecture and enable the integration of the new feature. This is necessary to ensure a stable and maintainable architecture, minimize architectural erosion and minimize integration effort of new features.

Use pair programming: After introduction into the feature teams, inexperienced developers should at first do some pair programming session together with an experienced developer. This enables the new developer to bridge the technological gap as well as allows easier transfer of knowledge about rules and standards.

Provide an infrastructure for documentation: Every project should provide an infrastructure to store and transfer knowledge, e.g. an open wiki. This is important to allow developers easy access to all necessary information to take part in the project.

Perform regular team meetings: The project team should meet on a regular basis. The authors argue that these meetings are a central practice and serve four major purposes:

1. They are a powerful management and planning instrument. Every developer should present the status of the current work upcoming development steps to keep track of the overall project status. Additionally new sprints are planned in these meetings.
2. The team meetings serve as a discussion platform for occurring problems in the design process. This has the benefits, that developers, who are stuck with a problem may find solutions there and it helps to spread technical information as well as process know-how.
3. The third purpose of those team meetings is to maintain the platform architecture. Results of architectural measurements should be discussed and architectural decisions be made. Also scoping sessions should be planed.
4. Maybe the most important purpose of team meetings is to create a team spirit and enhance developers' involvement in the project. Team meetings should thus have a casual ambience and encourage open discussions.

The next chapter explores how the presented strategies to face the challenges in large scale scientific software projects in terms of code, architecture and design process have been addressed in such a project. The CFD library OpenFOAM, which has been developed as a library for computational solutions in continuum mechanics with some of these specific challenges in mind will serve as an example.

5. OpenFOAM

OpenFOAM is an open source library, which offers functions and solvers for complex physical simulations. The Library is written in the C++ language and has been in development since 2004. It is being developed and supported by ESI-OpenCFD a subsidiary of SGI Corp (since 2011) which has been founded to support and develop the library.

The goal of OpenFOAM was to develop a flexible toolbox, which should solve some of the common problems, commercial simulation software is often plagued with. It has been developed specifically with these problems and challenges of developing large scale simulation software in mind and makes use of modern programming and software development techniques to address these challenges.

The construction of OpenFOAM makes heavy use of both, object oriented design and object oriented programming as it is offered by C++. It is one of the few scientific software projects, where the object oriented techniques have been used consistently. The toolbox has been developed with specific characteristics and challenges of large scale simulation software in mind. Thus OpenFOAM serves as a good example for the use of modern methods of software engineering within scientific software projects.

Jasak et al. [Jasak, 2007] have identified a set of common drawbacks of current simulation software, which they hope to have surpassed in the development of openFOAM:

- Simulation software is often very complex due to interaction between various physical models, different solution strategies and solver settings. This leads to bottle-necks in the development of the software as well as difficulties in testing and validation. In a monolithic simulation software, the development grinds to a halt as these overheads dominate the necessary effort.
- User requirements for simulation software are often general. While the developers might have certain applications in mind, these will not cover all requirements of later users. These requirements may involve experimental material properties, additional or new sets of equations and coupling with external tools and packages. Simulation software must thus be extensible.
- Every user will only use a certain subset of the functionality of a complex simulation tool. The other parts of the tool, will often still generate computational overhead, and certain make it more difficult for the user to understand the inner workings of the software. Simulation software must thus be modular and allow users to ignore and discard large parts of it.
- A more monolithic software incites developers and users to use identical discretisation and numeric mathematics even when they are clearly sub-optimal, simply because they "fit into the framework". Also economies of scale and price/performance for each new feature dictate which new models will be implemented, with a preference for established physics and thus discourage the development of new solution strategies since these then often require large changes in the software.

From this list it is clear that a monolithic general purpose software for simulation will never be able to cope with the full spectrum of simulations as the science evolves and new methods are developed, while it will be dragged down, by its exceeding size and complexity. These issues are not ones of mathematics or physical and numeric understanding, but stem from challenges in software engineering.
These points fit well to the two core challenges of scientific software, which where identified earlier, the highly changeable requirements and the high technical complexity of the software.

To address these challenges the developers of openFOAM have decided to fit the architecture of the library to these special requirements. With the last chapters in mind it should be clear, that application of modern software engineering techniques, tools and programming languages greatly enhances the chances of successfully addressing those issues.

5.1. Characteristics of OpenFOAM

The key strategy in the development of OpenFOAM is also strongly connected to the key characteristic of the OpenFOAM library. This is the consistent use of Object Orientation. Object Orientation is used as well for the design of the architecture as well as the internal structure of the code. The library also makes heavy use of several features of C++ which are only available with the use of object oriented programming. The authors of OpenFOAM claim, and support that claim with case studies that the consistent use of object oriented design combined with strict adherence to few coding practices makes it possible for OpenFOAM to be both, an efficient numeric toolbox as well as a flexible library, which fulfills the special requirements of simulation software [Jasak, 2007] .

The following pages present some of the most noteworthy attributes of the OpenFOAM library in regards to software engineering. Again these attributes will be separated into code, architecture and design process.

5.2. Code of OpenFOAM

Since OpenFOAM is implemented in C++ and not in a language, that was made explicitly for programming numeric software, strict rules were set up for the code. OpenFOAM makes strong use of some features of the C++ language for both efficiency as well as readability of the code. Since the project has become open source the use of these features is contained in the coding guidelines. These guidelines also contain strict rules on coding styles.

Object Oriented Programming:

Generics:

OpenFOAM makes heavy use of generic programming. Generic programming is a style of computer programming in which algorithms are written in terms of types which are specified later and that are only instantiated when needed for specific types. The data-type, which is actually used is chosen by the compiler during compile-time. This allows the compiler to automatically optimize type-specific implementations [Stroustrup, 2000]. In C++ generic programming is realized by the use of so called Templates.

The usage of generic programming and templates is not found very often in numeric code. The choice for heavy use of generic programming in OpenFOAM was deliberate to allow the code to be highly flexible and still allow easy testing for the algorithms. Generic programming using C++ templates allows for efficient reuse of algorithms, which depend only on some functionality of the used data type, like a comparison operator for sorting, and not on the inner workings of the data-type itself.
Examples for the use of templates and generic programming are Dirichlet boundary conditions, which function the same no matter if the field variable is a scalar, vector or tensor.

The use of templates enhances the flexibility and maintainability of the software without sacrificing efficiency and testability.

Inlining:

Object oriented programming is often said to produce code of lower performance, the cause of which is the execution time needed for class constructors and destructors as well as the overhead produced by a higher number of function calls compared to code with a more procedural design. These drawbacks are mitigated by the efficient architecture as well as the consistent use of the **inline** keyword for smaller functions in inner loops.

The inline keyword in C and C++ encourages the compiler to use inline expansion on the function which is declared with this keyword. Inline expansion is a powerful means for the compiler to optimize code for efficiency and the inline keyword can be used by the programmer to guide the compiler to functions, where inline expansion is suggested [Stroustrup, 2000]. Note that many compilers use inline expansion on functions without this keyword as well.

One of the benefits of inline expansion is that it eliminates overheads from function calls. Still more important is the fact that it is an enabling transformation as it transforms the code to enable the compiler to enact more automatic optimization on the code:

- Fixed constants, which are used as parameters are deleted by the compiler and thus dead code is eliminated.
- It allows to eliminate branches and keeps code which is executed together in close memory space to enhance instruction cache performance by improving locality of reference.
- The compiler might be able to reduce statements in cases, where only parts of the function are actually executed.

The drawback that it slightly enlarges the amount of machine code is not an issue in simulation software, where the memory requirements of grids, vectors and matrices outweigh those of the code by far.

Operator Overloading:

Operator overloading describes the possibility to use polymorphism to overload operators like “+”, “-” ,”=” etc.. Using this, the programmer is able to define these operations for different data-types like matrices ,particularly with regard to different sparse matrix storage schemes. While the internal working of these operators is different for each sparse matrix storage scheme the “look and feel” of the operations to their user stays the same [Jasak, 2007].

C++ allows to overload operators like +,-,= etc. for these more complex types. This greatly enhances readability of the numeric code as it follows the well known syntax of the underlying mathematical statements.

This is used consistently in OpenFOAM for numeric types like vectors, matrices and tensors. Operator overloading in these cases allows the programmer to largely use those data-types like he is used to operate on the underlying mathematical concepts. This enhances readability and understanding of the simulation code.

5.3. Architecture of OpenFOAM

The developers OpenFOAM put a strong focus on the libraries architecture to address the challenges faced in the development of simulation software. Their argument is that well engineered architectures and not clever coding techniques produce software which is able to fulfill the variable requirements for large scale simulation tools. The necessary flexibility to changing requirements demands this in particular. Jasak et al. [Jasak, 2007] argue that object oriented design of the architecture is what enables OpenFOAM to be flexible and still efficient.

This chapter will thus explore the architecture of OpenFOAM and show some more remarkable features of the architecture of the library as well as the design decisions given by Jasak et al. For these features.

Object Oriented Architecture

OpenFOAM does not only use object orientation as a programming paradigm to develop the source code of the library but also uses object oriented design principles to engineer the software's architecture. Object orientation is extensivly used in its basic form, where classes and object are used to mimic the behavior and relationship between the underlying concepts. The object oriented design in OpenFOAM thus forms a mapping from the domain model of the software (mathematics, physics, computational mechanics) to the architecture of the software. The object oriented approach allows the architecture of OpenFOAM to closely model the relationships and interaction between the concepts of the domain of numeric solvers and simulations. This forms a great advantage of object oriented design as it allows quite easy understanding of the code basis.

There are classes for data entities like vectors and matrices with the according operations as well as those for solvers, discretisation schemes etc. as well an abundance of others.

Open Architecture

OpenFOAM uses an open architecture, both in the sense of being open do further development through the open source community and open in the sense of an open framework/library. The following paragraph will explore the later as it is of more interest in regards to software engineering.

Frameworks and to a lesser degree libraries can be open or closed, which defines the way a user interacts with and modifies them. Closed frameworks and libraries only allow interaction in specified interfaces. The user only calls functions or creates classes of the library and does not do any direct modification. In the open case most if not all modifications are done using the classes of the framework/library itself. They are either used and modified directly or new classes inherit from them and introduce the individual functionality. Open architectures are thus much better suited to fulfill the requirements for simulation software stated in chapter two and three. They also again require use of object oriented design. The open architecture in OpenFOAM allows for easy and wide-scale customization while at the same time keeping the efficiency of the customized software by sticking to the underlying architecture of the solvers.

5.4. Design patterns in OpenFOAM:

Design patterns are general reusable solutions to a commonly occurring problems. They form descriptions or templates of how to solve problems which can be used in many different situations where problems of one family occur. Patterns are formalized best practices that the programmer must implement themselves in the application. They are a commonly used tool in object oriented software construction and allow developers to easily construct architectures for given problems. The first comprehensive list of patterns was published by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides [Gamma, 94] and there is a large number of patterns known by now. Observing some of these patterns in the architecture of OpenFOAM is useful to show the use of modern object oriented design in a simulation software.

The use of some design patterns can be recognized in the architecture of openFOAM. Although it is unclear, if this use was planned or came naturally since there is a lack of design documentation. If not noted otherwise all exemplary design patterns where extracted from the OpenFOAM documentation which is generated from the code base using doxygen [http://www.openfoam.org/docs/cpp/]

Strategy Pattern: lduMatrix::solver Class

The strategy design pattern defines a family of algorithms, encapsulates each one and makes them interchangeable to the outside user. The user is then able to change the best "strategy" out of a set for a given problem.

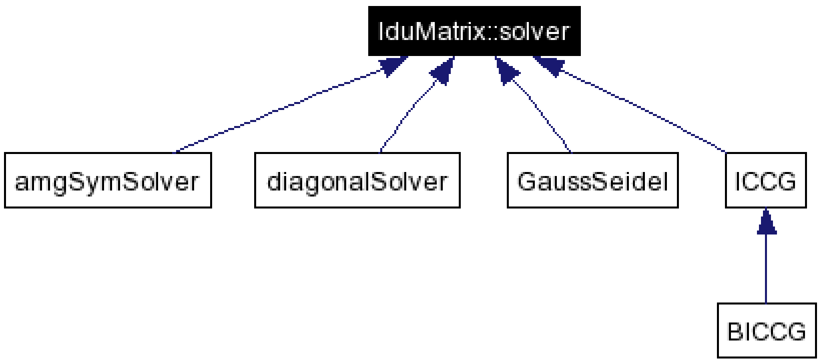

Figure 3: Inheritence structure of matrix solvers

The Class lduMatrix::Solver forms a nice case of the *Strategy* design pattern. The class offers functionality to solve linear equation systems with a matrix of class lduMatrix (which is the standard, most general matrix class in OpenFOAM). All solvers are classes, which inherit from the general class lduMatrix::solver. Every Class, which uses a solver to solve a linear equation system uses a child of lduMatrix::solver and the actual solver used is chosen according to the properties of the matrix. This design pattern shows the advantage of object oriented design as the code structure follows the understanding of the behavior and use of numeric solvers for linear equation systems.

Template Method Pattern: Since OpenFOAM makes such heavy use of generic programming it comes to no surprise, that there are many cases of the template design pattern. One example for this is gradient calculation with the class gradScheme, which supplies basic functionality of gradient calculation as a template method and is then extended and implemented in a variety of classes, where each implement a concrete algorithm to calculate gradients.

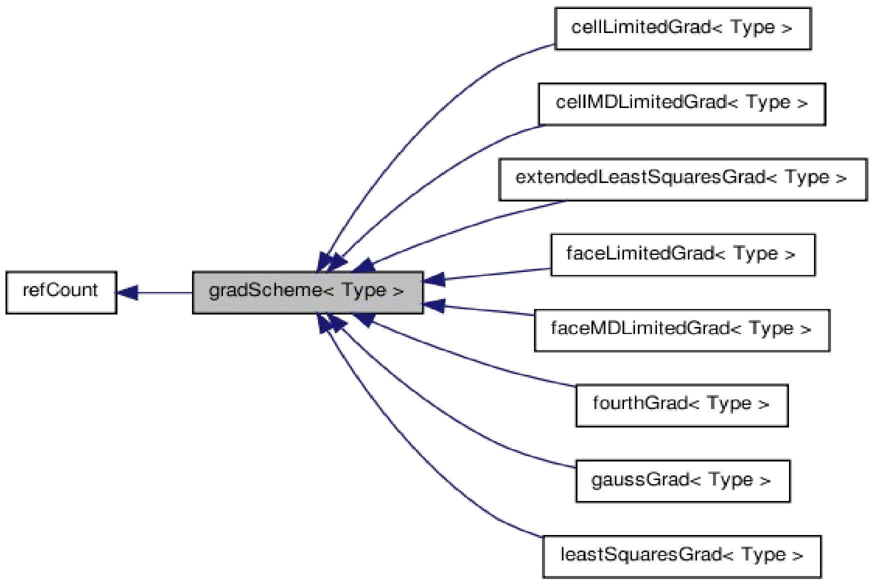

Figure 4: Inhertince structure and template for gradient calculation

There are certainly more design patterns to be found in the code base of OpenFOAM but the size of the library and a lack of design documentation makes it unfeasible to construct an exhaustive list of those in this work.

5.5. Design Process

In its current state as an open source software, OpenFOAM uses a design process typical for open source projects. Unfortunately there is no literature or other information available about the design process prior to the change to an open source platform.

In the current phase of the design process of OpenFOAM seems to consist of the rules and principles employed by many open source projects. There is no strict process model given but a set of rules and a close inner group of people which check code contributions for compliance to these rules and general quality. Nonetheless this process shows some similarity to the best practices suggested by Hoffman and Lichter as well as Baxter et al. [Baxter, 2006]. The lack of a consistent design process model also shows in some problems OpenFOAM does suffer from, like a lack of design documentation.

During its long development process the documentation of OpenFOAM has not been very extensive. Most documentation is in form of doxygen generated documents, which stem from code documentation. Especially design documentation and the rationales behind design decisions of the architecture are not visible to users and current collaborators.

This indicates a lack of an explicit design process model for the development of OpenFOAM. Such a process model, should allocate resources to create, maintain and distribute the different artifacts of documentation of OpenFOAM.

Process of Code Contribution:

Contributors upload their contributions (source code, in all but the rarest cases) to the unsupported contributions repository. There all users of OpenFOAM may share their code to the wider community but it is not part of the official OpenFOAM distribution.

In irregular intervals some of these contributions are checked for validity and adherence to the coding guidelines by the core team, which is supplied by ESI-OpenCFD. If the contributions meet the standards they may then be migrated to the official distribution of the library. Other parts of the library are developed by ESI-OpenCFD itself and then made part of the next release. Features implemented by ESI-OpenCFD are often added by request from users of OpenFOAM. Unfortunately there is no information available about the process of how these feature requests are handled internally.

6. Conclusion and Outlook

Scientists in the disciplines of simulation, numeric mathematics and computational mechanics spend increasingly effort and resources in developing software. These software projects often reach dimensions, where the problems arising from the software development process form the actual bottle-neck of the scientific progress. These problems of developing large scale simulation software should thus be addressed in a quantifiable and systematic way evolving the development of software into engineering of software.

This paper shows that scientific software projects and simulation software projects in particular face challenges beyond those regularity faced in common commercial software. These challenges need to be addressed in the software's code, architecture and design process.

Some basic coding techniques are presented, which enable simulation software to fulfill its requirements in efficiency. The most prevalent of them is efficient cache use.

Object oriented design of architectures and the architectures, which result from this design paradigm are presented as an effective way to address some of the challenges scientific software projects and simulation software projects face.

A preliminary design process model for the development of scientific software is described. This model takes the particular challenges of developing scientific software in account and promises to tackle these challenges through the combination of an agile approach and stringent application of rules to keep the process manageable.

The simulation toolbox openFOAM has been explored as an example of a large scale simulation software. In the last pages it was shown how some of these challenges where addressed during the development process of openFOAM. OpenFOAM does make use of modern programming paradigms and enforces an efficient and stringent architecture. This enables it to be an flexible and efficient while still large example of simulation software. It uses the object oriented programming paradigm to its fullest extend. Modern architecture features like templates and design patterns can be found in its code structure. OpenFOAM does not (yet) make use of particular design processes or project management techniques to enhance the development process. It nonetheless has some rule which are enforced and mechanisms of how new features are added to the library and how changes are managed which come from common development processes of the open source community. These rules and mechanisms are also useful for the development of scientific software and simulation software.

This leads to the conclusion that modern software engineering techniques like object oriented programming and the use of modern programming languages and object oriented design of system architectures with the use of well known best practices like design patterns. These tools have to be combined with stringent project management and should be supported by a dedicated design process model for large scale scientific software projects. The example of OpenFOAM shows that this concept is feasible and provides substantial advantages to the development of simulation software.

While the requirements for simulation and numeric code are well known and many solution strategies exist, those requirements on the architecture and design process of simulation software have so far been largely neglected by the software engineering community. Those scientists working in the field of simulation software are still lacking the proper tools which are widely used by commercial software developers around the wold. The presented works propose some new tools to introduce into this field of research, but are far from exhaustive enough to provide the wide range of scientific software development with the necessary tools so that these can one again concentrate on their actual scientific subjects.

Thus a lot of further research is necessary to develop architecture frameworks and design processes for scientific software projects in general and large scale simulation projects in particular.

Bibliography

[Wilson, 2006] Wilson GV (2005) *Where's the real bottleneck in scientific computing?* Am Sci 94:5.

[NATO, 68] (1968) *SOFTWARE ENGINEERING Report on a conference sponsored by the NATO SCIENCE COMMITTEE* Scientific Affairs Division, NATO

[Hoffmann, Lichter, 2011] Nyßen, A., Lichter, H., Hoffmann V. (2011) *Processes and Practices for Quality Scientifc Software Projects*

[Jasak, 2007] Jasak H., Jemcov A., Tukovic Z. (2007) *OpenFOAM: A C++ Library for Complex Physics Simulations* International Workshop on Coupled Methods in Numerical Dynamics

[EJB, 2013] Enterprise Java Beans, (2013) Oracle – project website.

URL: http://www.oracle.com/technetwork/java/javaee/ejb/index.html

[Baxter, 2006] Baxter S., Day S., Fetrow J., Reisinger S. (2006) S*cientific Software Development Is Not an Oxymoron* PloS COMPUTATIONAL BIOLOGY 2:9

[Oliveira, 2006] Oliverira S., Steward D. (2006) *Writing Scientific Software* Cambridge University Press

[Hello2Morrow, 2010] Hello2Morrow, (2010). Sotoarc - project website.

URL: http://www.hello2morrow.com/products/sotoarc

[Metrics Project, 2010] Metrics Project, (2010). Metrics - project website.

URL http://metrics.sourceforge.net/

[Stroustrup, 2000] Stroustrup B., (2000) *The C++ Programming Language* Pearson Education Inc. 3. Edition

[Gamma, 94] Gamma E., Helm R., Johnson R., Vlissides J., (1994) *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley

Table of Figures