Transport in Porous Media
https://doi.org/10.1007/s11242-019-01310-1

Development of Open-Source Porous Media Simulators: Principles and Experiences

Lars Bilke[1] · Bernd Flemisch[2] · Thomas Kalbacher[1] · Olaf Kolditz[1,3] · Rainer Helmig[2] · Thomas Nagel[1,4]

Received: 21 December 2018 / Accepted: 17 June 2019

Abstract
By its very nature, research into multi-physical processes occurring in porous and fractured media requires a collaborative approach. An interdisciplinary approach has led to the adoption of collaborative software development paradigms in this field relying on software for scientific computing as research infrastructures. The development of open-source software has become a cornerstone of computational approaches in academia and has even spawned successful business models in the commercial world. This article is geared toward readers who want to learn more about potential benefits of open-source software in porous media research and who want to familiarize themselves with typical workflows required to become an active contributor to or user of open-source solutions for porous media simulation. The article puts general principles, motivations and concepts into the specific context of experiences and lessons learned from the authors developing the open-source software projects OpenGeoSys and DuMu^x.

Keywords Open-source · Multi-physics · THMC · Software development · Porous media

1 Introduction

As in virtually all fields of science and engineering, computational modeling has become ubiquitous in porous media research and serves a large number of indispensable functions.

Submitted to the Special Issue in Celebration of InterPore's 10th Anniversary.

✉ Lars Bilke
lars.bilke@ufz.de

1 Department of Environmental Informatics, Helmholtz Centre for Environmental Research — UFZ, Leipzig, Germany

2 Department of Hydromechanics and Modelling of Hydrosystems, University of Stuttgart, Stuttgart, Germany

3 Applied Environmental Systems Analysis, Technische Universität Dresden, Dresden, Germany

4 Chair of Soil Mechanics and Foundation Engineering, Geotechnical Institute, Technische Universität Bergakademie Freiberg, Freiberg, Germany

Published online: 25 June 2019

The computational models typically encountered in porous media science are strongly rooted in physical theories and thus enable the interpretation of experiments and the rationalization/condensation of large amounts of data in terms of basic governing principles, typically composed of balance laws and constitutive relations (Diersch 1984; Gray and Hassanizadeh 1998; Helmig et al. 1997; Ehlers 2002; Hutter and Jöhnk 2004; Coussy 2010). By extension, they make accessible quantities which are difficult or impossible to measure in an experiment but are of high practical interest. Computational models support the design, engineering and rapid development of technical systems in iterative (i.e., circular) workflows spanning the steps development, verification, calibration, validation and prediction (Oberkampf et al. 2004; Thacker et al. 2004). Their educational value lies in their fast, cheap and, above all, interactive access in the sense of "experimentation with theory," providing immediate feedback to the student regarding effects of altered boundary conditions, material properties or numerical settings. Researchers on the other hand can test certain hypotheses directly *in silico* before returning to physical experiments. This can become fundamental in the exploration of phenomena which are prohibitive to perform experiments on—either for ethical or for technical reasons—especially in safety-critical (e.g., nuclear industry), controversial (e.g., biomedical research) or extremely large-scale settings (e.g., geo-engineering). Finally, in conjunction with scientific visualization, computational models pose value for communication with political stakeholders or management, in that they allow the comparative assessment of scenarios and can help communicate complex scientific/engineering concepts to people not trained in the field.

Research in porous media is often concerned with multi-physical phenomena: fluid and solid phases interact in diverse ways (consider as examples multiphase flow, particle transport and deposition, phase transitions, etc.); deformation processes change the pore space and interact with fluid flow (fluid-solid interaction); chemical reactions involving the transport of dissolved species occur in the presence of large reactive (inner) surface areas and can likewise alter the pore space; heat transport is a combined effect of conduction, convection/advection, interphase transfer, etc.; biofilms form on solid interfaces or can traverse pores, again changing the properties of the porous medium and the composition of the permeating fluids. For an exemplary impression of some of the discussed phenomena, consider the following articles and references therein: Helmig et al. (1997), Huyghe and Janssen (1997), Gray and Hassanizadeh (1998), Kolditz (2002), Ambrosi et al. (2011), Diersch (2014), Ehlers and Häberle (2016), Nagel et al. (2016), Carrel et al. (2018).

This list is by far not exhaustive, but it illustrates the crucial point: In many applications of practical relevance, these phenomena[1] do not occur in isolation, but are coupled, i.e., they interact. Their modeling requires knowledge from an equally diverse range of scientific disciplines, numerical mathematics and computer science. The need for a collaborative research environment becomes apparent from this description, which in turn can benefit greatly from jointly maintained research infrastructure. For computational modeling, aside from the designated hardware, software for scientific computing can be considered such an infrastructure.

This software infrastructure has to adapt to the evolution of scientific questions and engineering demand. With the development of tailored solutions and the integration of knowledge from other disciplines, scientists require the ability to directly implement novel physical models or adapt existing numerical implementations to their needs by themselves. This often involves taking an exploratory approach to phenomena not yet fully understood. One solution to this challenge is to involve the scientists in the development process of their numerical

[1] Often called thermo-hydro-mechanical/chemical-biological coupled problems, THM/CB.

tools, support them with expertise from information sciences and eventually provide them with exactly the software environment they need. This leads to a fast-paced development requiring agile software development paradigms that are able to keep up with the scientific progress. In addition to this development perspective, the scientific method centers around concepts such as reproducibility. In the present context, this means that the process from input data over models and simulations to scientific results should be fully transparent and reproducible. This has led to the requirements of open data, open source and open development processes. Finally, and from a more technical perspective, many computational models reach a scale that necessitates the use of high-performance computing infrastructure, which places specific constraints on software architecture and licensing schemes.

This paper highlights the development of open-source software for scientific computing with a particular reference to multi-physical processes occurring in porous and fractured media. We tried to focus on those benefits and features of an open software approach that can to some degree be generalized to the class of projects of interest in porous media research.

Therefore, this paper aims to

- depict the principles behind and the potential benefits of open-source software and community-driven development;
- give a compact overview of an easy-to-read introduction to the various aspects scientists with the ambition to get involved with software development should familiarize themselves with (platforms, licenses, workflows);
- substantiate these principles and benefits as well as exemplify these various aspects by means of two software projects coordinated by the authors;
- inspire the reader to get involved and contribute to a growing community as well as to raise awareness that open source does not mean free of charge and that there is a need for long-term development and funding.

In terms of specific packages, we will concentrate on experiences made with DuMu^x, dumux.org and OpenGeoSys, opengeosys.org, being the two projects that the authors are coordinating and co-developing. Obviously, there are many other open-source endeavors focusing on Darcy-scale porous medium flow and transport processes, such as FEBio, febio.org (Maas et al. 2012), MODFLOW, water.usgs.gov/ogw/modflow/ (McDonald and Harbaugh 2003), MRST, sintef.no/projectweb/mrst/ (Lie 2019), OPM, opm-project.org (Baxendale et al. 2018), ParFlow, parflow.org (Maxwell et al. 2015), PFloTran, pflotran.org (Lichtner et al. 2015) or PorePy, github.com/pmgbergen/porepy (Keilegavlen et al. 2017). Another possibility is to set up on an open-source package with a broader focus such as deal.II, dealii.org (Bangerth et al. 2007), DUNE, dune-project.org (Bastian et al. 2008), Feel++, feelpp.org (Prud'Homme et al. 2012), FEniCS, fenicsproject.org (Logg et al. 2012), MOOSE, mooseframework.org (Gaston et al. 2009) or OpenCMISS, opencmiss.org (Bradley et al. 2011).

In the above we took a development-oriented perspective geared toward open-source developments. Although we will not address alternative approaches in detail, we at least want to briefly mention that there are many settings in which other, e.g., commercial closed-source, concepts are successfully employed. Such application-oriented software is especially attractive in the context of well-defined problems for which a robust, stable and established environment with clear responsibilities of the software provider is sought and which is accepted by key industrial partners. Adaptation of these tools by the users is possible to certain degrees via specific interfaces (e.g., user materials, user elements, user defined boundary conditions, etc.) which is entirely sufficient for many applications. Core functionalities can, however, often not be accessed directly and their verification can become involved (Ji et al.

2013; Bazant et al. 2012; Görke et al. 2010). The application-oriented focus and commercial nature of these software solutions also mean they are often complementary to open-source solutions both in terms of strengths and weaknesses.

The article is organized as follows: In Sect. 2, the expected benefits of developing open-source software will be described along with some key definitions, a brief discussion on licenses and a description of community-building measures. Section 3 familiarizes the reader with basic technical infrastructure and workflow concepts which may be necessary to start contributing to or even just benefiting from software developed by a community of researchers. While Sects. 2 and 3 are of rather generic nature, the principles and benefits will be substantiated and the concepts will be exemplified based on the authors' first-hand experience in Sect. 4. The paper closes with specific lessons learned and some general conclusions in Sect. 5.

2 Expected Benefits and Principles of Open-Source Software in Research

While free and open-source software (FOSS) is ubiquitous in many people's lives, it is still not common for many academic individuals and groups to develop and release their in-house code in an open-source manner. In this section, we first provide a motivation for developing open-source software in Sect. 2.1 and we discuss briefly FOSS definitions and licenses in Sect. 2.2. Then, we highlight the importance of establishing a sustainable community around an open-source project in Sect. 2.3.

2.1 Why Develop Open-Source Software in Academia?

In the following, we give three arguments for developing open-source research code as an individual researcher or academic group. The first states that providing access to source code is mandatory as reproducibility is fundamental to the scientific method. The second argument is the expected increase in code quality and applicability. Finally, the potential for collaboration with industrial or academic partners is greatly facilitated and simplified by using open-source development principles. Obviously, many other works are dealing with potential and realized benefits of open-source development, with Raymond (1999) being probably one of the most influential ones.

In their "Proposals for safeguarding good scientific practice" (Forschungsgemeinschaft 1998), the German Research Foundation DFG rephrases one of the main ingredients of the scientific method: "The primary test of a scientific discovery is its reproducibility." The European Commission advices members on installing a wide range of Open Access policies to "…enable the use and re-use of scientific research results" (European Commission 2018). The increasing complexity of scientific results renders their reproducibility a highly challenging task. The only viable way of enabling the scientific community to reproduce the results gained by computer code is to grant access to this code as well as to the input files and data. The scientific field of reproducible research is concerned with all correspondingly related aspects (Fomel and Claerbout 2009; Kitzes et al. 2017). While we don't investigate this any further here, we acknowledge the fact that source code is an indispensable ingredient of any reproducible computational result or the description/evaluation of such a result.

Another motivation for developing academic research software by means of open-source principles is the expectation that code quality will increase and the period of vocational

adjustment for new users and developers will be decreased. Allowing the community to look at the code and to use it for any purpose will increase the number of opportunities for detecting programming errors and limits of the proposed numerical model or algorithm. User feedback will also help to improve the quality of the code, including the coding style or the number and comprehensibility of code comments. If an active user community emerges, it can also be expected that "programmers who report problems with an open-source project often not only provide a problem description but also contribute a software patch which solves the problem" (Puder 2000). Overall, an open-source development model can improve code quality, readability, applicability and robustness. This in turn is highly beneficial for the academic group developing the software and will allow a sustainable and long-lasting program development that improves the usually encountered "development model" in terms of highly specialized fragmented programs where the life span is the duration of employment of the corresponding individual PhD candidate or postdoc.

Open-source code development can also attract industrial partners as well as foster collaboration in joint research projects between academic institutions. The expectation is that partners and collaborators profit from an accelerated technology transfer, since new research results will be immediately available as free software. The development, testing and exchange of new models and computational methods will be enhanced and simplified. While commercial simulators may only be suited to cope with non-standard model or parameter descriptions to the often limited extent that suitable programmable interfaces are provided, dedicated FOSS solutions might directly offer such descriptions or can at least be enhanced to do so. While all arguments have been discipline-agnostic so far, we now also specialize a bit more in direction of porous media research. Especially in safety-critical applications like the use and protection of the subsurface, the answer of only one simulator is not enough to build confidence in the computed results. To this end, joint benchmarking such as described, proposed and undertaken in, for example, Segol (1994), Islam and Sepehrnoori (2013), Class et al. (2009), Flemisch et al. (2017), Kolditz et al. (2018b) is an invaluable measure. Open-source code not only offers the opportunity to calculate and compare the results from different simulators but also to verify and validate their implementations, helping to increase the confidence of modelers and decision makers. Obviously, it needs much more for a FOSS solution to be integrated into daily industrial workflows by replacing a commercial simulator: ease of use, support of industry standards for input data and a wide range of options to run interesting cases. In reservoir engineering, OPM (Baxendale et al. 2018) and MRST (Lie et al. 2012; Lie 2019) are examples for projects that are to a large extent dedicated to cope with corresponding industrial requirements.

It is debatable whether the aforementioned benefits can only be achieved by following the open-source idea, and, even more, whether these benefits can really be achieved and are not only wishful thinking. In Fuggetta (2003), the author dissociates himself from the belief that only open source can provide the desired advantages and that it is *the* solution to many problems. However, he makes a crucial exception for research codes: "Nevertheless, there are particular market situations where open source is probably the only viable solution to support successful and effective software development. Typical examples are research communities that need specific software products to support their research work."

2.2 Definitions and Licenses

Free and open-source software (FOSS) has a history that is as long as software itself, see González-Barahona et al. (2013) and the references therein. There are currently two

main organizations behind the FOSS idea: the Free Software Foundation (FSF), fsf.org, and the Open Source Initiative (OSI), opensource.org. While the FSF maintains the *Free Software Definition*, gnu.org/philosophy/free-sw.html, the OSI stands behind the *Open Source Definition*, opensource.org/docs/osd/. Although the two organizations have fundamentally different motivational backgrounds, both "lead to the same result in practice," and the terms *free software* and *open-source software* "are essentially interchangeable," as stated at Open Source Initiative (2018). In the following, it will not be distinguished between the two terms, and usually the term "open source" is used.

For the developers of open-source software, choosing the right license can be a crucially important step. This step is especially complicated due to the availability of currently more than 70 OSI- and FSF compliant licenses, many of which only differ in small details which possibly have important legal implications. A good overview of the main possibilities is provided in Morin et al. (2012). In general, one should use one of the licenses which are approved both by the FSF and the OSI. It is also advisable to choose a license from among the most common ones that are used for many other open-source projects. If the developed code is based on another project, the best choice usually is to stick with the license from that particular project. The two most fundamental differences that an open-source license can make in practice are the following:

- Allowing that the code can be linked with code/programs distributed under another license. This can be important for the further utilization of the code as part of an otherwise proprietary program.
- Requiring that changes to the code have to be released under the same license. Such licenses are referred to as "copyleft" type licenses (Stallman 2018).

It remains very difficult to recommend one particular license. For example, DuMu^{x}is issued under the most traditional copyleft-type license GNU-GPL (version 2.0 or later), the "GNU General Public License" (GPL 1991), while OpenGeoSys is distributed by means of a permissive BSD-style license (BSD 2018). In Stodden (2009), the author argues that computational research produces an entire research compendium which comprises the standard publication, the data, the computational experiment including the source code and any auxiliary material. She proposes to release the entire compendium under a "Reproducible Research Standard" (RRS) that builds upon common licenses for the mentioned components. While it seems that the RRS hasn't been adopted yet, she strongly argues for attaching a permissive license to the code by means of the modified BSD license. One main problem with choosing a license is that a later change can be a very difficult step. Theoretically, everyone that owns copyright on a piece of the code involved has to agree to such a change. This might be complicated, if such persons already left the development team or if someone disagrees with the proposed change, etc.

2.3 Transferring an Open-Source Project to an Open-Source Community

Creating a sustainable community around an open-source project is a challenging task because of constant change in both software development and science but also in personnel (viz. typical PhD or postdoc cycles). A certain critical mass of contributors who support both the development and the application of the software is required to keep the momentum of the project. The addressed research applications and questions can be very different between members of the community, which often makes coherent community efforts less obvious, but also highly rewarding once they are established. In the early stages of a project, an initial focus on a particular scientific topic can certainly be helpful. However, in general, the bigger

the community, the more scientific topics can be tackled, and more synergies can be created (if people can still agree on the overarching direction of the project). Many core features, such as numerical solution methods, boundary condition types or coupling strategies, can be cast into generalized concepts in terms of both theory and software to benefit a wider range of applications.

Yet, the application of the software is not just about the numerical core and its performance but also about robust simulation workflows as well as a skilled and connected user base. Over the years, ecosystems have evolved around the various codes. Such software ecosystems provide the means for integrating data and information into models, setting up simulations in view of the habits of the different scientific domains in which they were implemented, and providing methods, tools and workflows for the evaluation and interpretation of results. In a wider sense, such ecosystems facilitate the exchange of information, resources and concepts between scientific institutions, government agencies and companies. The result is a research infrastructure that provides the foundation for scientific activities and cooperation. Strong communities are international, as well as interdisciplinary and intergenerational.

One of the most important things a community needs is a solid communication platform. In the recent past and until today, the most popular methods are (publicly archived) mailing lists and forums. Communication via code hosting platforms (e.g., GitHub or GitLab) has become common too at least for development-related issues. Experience has shown that it is not worthwhile to consider too many different options of communication as the information gets scattered and it can become confusing to recover past communication. It should go without mention that a welcoming and friendly atmosphere should be created: people should feel able to contribute at all times, every contribution, whether a small feature or a large extension, is valuable and no question should be too trivial to be answered. Community meetings organized regularly help in highlighting the diversity of the scientific topics often only seen by the core development teams, discussion of new issues and code changes associated with major changes (e.g., basic interfaces) and simply in creating the mentioned connected user base. Finally, it is important to acknowledge the contribution of developers not working directly on the scientific projects by actively integrating them in the project teams.

3 Workflows in Open-Source Software Development

In general, scientific software development highly differs to common software development as requirements are not known in advance, new hypotheses emerge during development and often the scientist is also the developer, tester and user of the software in one person (at least in small- to medium-sized software projects). Scientific software development therefore needs an incremental planning model with changing specifications as the scientist reacts on unforeseen research directions (Segal and Morris 2008). It is critical that the scientist's ever-changing development requirements do not impair goals and implementations of other involved scientists thus demanding an efficient development and quality assurance (QA) process which gives instant and in-depth feedback. In the past, software engineering best practices were often neglected in scientific software projects (see Kelly (2007) and Kelly and Sanders (2008)). Because of software growth in both size (lines of code) and complexity, projects must adopt as described in Wilson et al. (2014) and Kempf and Koch (2017). After listing and discussing infrastructure components for open-source projects in Sect. 3.1, we will outline a typical open-source development workflow in Sect. 3.2. We employ a rather

general perspective which will be complemented in Sect. 4 by the specific choices undertaken in DuMu^{x}and OpenGeoSys and the resulting experiences.

3.1 Infrastructure for Open-Source Projects

In the following, several important parts of the infrastructure of an open-source project are described briefly. All of these parts can be realized by open-source solutions themselves. In particular, code hosting, revision control, website, bug tracking, automated testing, mailing list and project analysis tools are discussed.

Revision control Revision (or version) control is the management of changes to the program code and accompanying files like documentation, build system, etc. Corresponding tools are indispensable for joint code development (and at least very helpful for individual development). They can be classified into two categories: centralized and distributed. The classical centralized approach with an authoritative server-client model plays no important role anymore. In distributed revision control, there is technically no single authoritative repository. All individual working copies themselves are repositories, and data access/changes can be done referring to each one of those. Nevertheless, a community has to agree on an accepted authoritative repository. In recent years, Git, git-scm.com, has emerged as the most widely used distributed revision control system. With Git LFS (Large File Storage, a Git extension), git-lfs.github.com, it is even possible to efficiently versionize large binary files. Distributed systems share many advantages over centralized systems. Almost all operations (only exception are synchronization operations to other repositories) on distributed systems can happen on the local computer system (thus aiding a good performance) and without Internet access. Distributed systems allow arbitrary flexible workflows although they have to be clearly defined within a development community.

Code hosting A central place to store the agreed on authoritative repository (or the centralized repository itself) serves both as an access point for developers and users and as a reference location for releases. In academic environments, such a place may be provided on the servers that are administrated by the academic institution itself. Nevertheless, it can be beneficial to make use of one of the available code hosting services like GitHub, github.com, BitBucket, bitbucket.org, or GitLab, gitlab.com. They do not only offer to host the code, but also several of the additionally needed infrastructure components like website, bug trackers and discussion boards. GitLab is open-source itself and can be self-hosted to minimize the dependency on third-party services or for ensuring data ownership and privacy.

Code publication In view of reproducibility, it is desirable to be able to refer to particular instances of a code basis. For example, if the results described in a scientific article have been achieved with one such instance, this should be cited properly in the article. While this is technically easy with Git by creating/using a tag with its corresponding commit hash, such measures don't provide persistent identifiers. In order to overcome this, services like Zenodo issue, for example, DOIs for uploaded data sets, zenodo.org. In this case, the data set can consist of an archive file of the repository at the desired instance, which usually is easily obtainable from the version control management system. As a positive side effect, a list of authors can and has to be attributed with the data set. In academic environments, the author list of a scientific paper describing a code basis soon becomes outdated in the sense that a citation of only that particular paper would disregard recent contributions and contributors.

Being able to cite a particular instance of a code allows for acknowledging the work of the actually involved persons.

Website A dedicated website is the showcase of the developed software for the outside world. It should be easy to find and attract potential users to download and test the code. It should provide access to the source code and, if available, to binaries or packages for different platforms and example data (e.g., for benchmark cases). It should also contain basic installation and usage instructions and pointers to the detailed documentation for e.g., the code, coding standards, information on how to contribute, licensing policy and additional documents like handbooks or tutorials. To illustrate the capabilities of the simulator, a gallery with screen shots and illustrations from selected applications and a list of achieved publications and funding grants or projects as well as detailed feature lists are helpful.

Mailing list and discussion forum Structured and regular information by mailing lists can help to interact with the user community of a software project. Anybody interested can sign up for the list and post and receive emails dedicated to the project. From the developers' side, the list can be employed to make important announcements like the release of a new version or the detection of a severe issue. The users should be encouraged to post questions related to the code that can be answered by the developers or also by other users. Archiving the mails helps to keep track of the topics discussed and people can respond to posts whenever they want, no matter what time zone they are in. Modern approaches combine mailing lists with discussion forum functionality such as the Discource project, discourse.org.

Issue tracking When developing software, it is important to keep track of software bugs and other issues like feature requests. An issue tracking system is a tool that helps to achieve this bookkeeping by means of a database. A large variety of such tools, many of them free of charge and with an easy-to-use web interface that can be customized to the project at hand, exist. Code hosting services such as GitLab, BitBucket and GitHub integrate issue tracking tools with the hosted code. Issue tracking systems help to administer and to keep track of this information and to resolve the issues. While having such a tool is helpful for any kind of software and development team, the open-source approach enables the users to file high-quality bug reports and to assist the developers in improving the code quality. The users themselves can find out the precise location of the bug and thus perform the first important step toward its removal. It is also possible that a user resolves the issue himself and provides a patch for the developers that can be directly applied to the code base. Other users can instantly profit from the improved software.

Automated testing and dashboards Writing and performing tests is another indispensable ingredient of sustainable software development. Since performing such tests manually can be a time-consuming, cumbersome and error-prone task, it is desirable to use tools for test automation. A large variety of such tools facilitate the definition, building and execution of a test suite. Widely used platforms for automated testing and dashboards are Jenkins, jenkins.io, Travis, travis-ci.com and BuildBot, buildbot.net. The tests can then be performed automatically, for example at fixed time intervals or whenever a change to the code has occurred. Depending on the individual outcome, each test can then be marked as passed or failed. Since the amount of data created by building and execution of the test suite can be very large, it is highly desirable that the automation tool also provides the possibility of a suitable visualization or a textual summary of the test results and potentially detected issues.

All of this information can be published as a dashboard website to help the user navigating the issues.

Project analysis It is a challenging task to measure the quality of an open-source project, apart from subjective user opinions. Some websites like Open Hub, openhub.net, "a free, public directory of free and open-source software" offer this service. If a project is registered that offers anonymous read access to its source code repository, the code's history and ongoing updates are analyzed. It provides measures like the number of commits and committers to the project, what programming languages are used and how the code lines are distributed among actual code, comments and blanks. Based on this and the evolution of these numbers, labels are given that provide information about, for example, the size and activity of the development team, as well as the stability of the development activity. Moreover, it provides an estimate for the cost of the project based on the Constructive Cost Model Boehm (1981). The provided statistics can be valuable to developers of the project itself as well as to third parties. For instance, the developers can reflect if the numbers meet their expectations and possibly think about counteractions if, for example, the development activity decreases. Moreover, the provided measures can help interested parties to decide whether to use the software or to contribute to the project. Finally, potential sponsors can employ them to judge if it is worth to fund the project.

3.2 Development Workflow

This section outlines the typical development workflow starting at setting up a development environment over code contribution to code maintenance.

Setting up a development environment Every open-source project should provide clear and simple instructions for new users on how to set up a development environment on the user's machine. These instructions are often given in the form of a README-file directly as part of the source code. For more complex set up instructions up-to-date information is usually provided at the project website. Typical instructions contain information on how to obtain the source code from the revision management system, how to install required other software (e.g., compilers, code libraries), how to configure the software (i.e., defining in which configuration the software will be built) and how to generate the application binary. At the end the user should be able to edit the source code, eventually compile it into an executable binary (not necessary for scripting languages) and conduct computations using the newly added functionality.

Contributing code To provide software to a diverse scientific community under common quality standards, a development workflow should be established that requires a well-defined task sharing between core developers (inner ring), application-oriented developers (middle ring) and users (outer ring) (Fig. 1). A workflow driven simultaneously by both sides is able to include both application and development perspectives. Application orientation guarantees the practical relevance of code development and the definition of meaningful test cases (application benchmarks). Core development is ideally conducted by experts in software engineering and information science having a broad scientific and strategic overview, such that core development in large software projects can guarantee a high level in software quality and flexibility for a variety of applications in the research field. This flexibility is intended to ensure both rather straightforward extensions in functionality and in particular a rapid implementation. The middle ring "application-oriented development" is of crucial importance in several regards, e.g., for translating the science cases to the developer level and transfer-

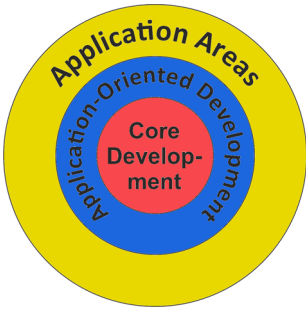

Fig. 1 Illustration of a core developer/user perspective on collaborative software development for flexible simulation platforms

ring the corresponding technical conditions back to the users as well as for communication between core developers (information scientists) and users (domain scientists).

Code should be written in respect to any given guidelines such as for code style and structure, code and feature documentation as well as code testing. New functionality should be supported by new software tests which verify the implementation. To some degree this can be done locally as the user develops the functionality. Complete guideline conformance may involve a more sophisticated test set up incorporating multiple computing platforms, a magnitude of software configurations and maybe CPU intensive complex tests. All these test configurations should be verified by automated tests which are often also automatically set up and executed on dedicated hardware when the user proposes the code change on the code management platform (e.g., the authoritative code repository). Once the verification tests (e.g., tests against numerical solutions or experimental results) are complete the user should be notified about the results. If issues were raised, the user has to rework their implementation until no remaining issues were detected in subsequent test runs. Automatic tests are very helpful in detecting common errors which can be checked by a computer program such as compile errors or missing implementation documentation.

But for a sustainable development contextual feedback by core developers of the open-source project is crucial to maintain a consistent development style and to follow the community-agreed general direction of the software project as a whole. "Code review is systematic examination ... of computer source code" (Wikipedia 2018) and often involves at least one of the software's core developers and the user. The core developer checks the proposed change for general acceptance and may give hints for improvement (of, e.g., the computational performance, the code structure, the general guidelines conformance). Other domain scientist should step in the review process to verify the newly implemented tests from a scientific perspective. Once the iterative feedback loop between the user, code reviewer(s) and automated test system satisfies all aspects the proposed change is merged into the codes main development line on the authoritative repository.

Maintaining code There should be at least a small group of core developers or maintainers who are able to work on the project on a regular basis to fulfill a variety of recurring tasks: giving assistance and feedback to other developers either in person or by means of code reviews on the project management platform; plan and implement larger code refactorings; maintain the code with respect to new coding standards, principles and best practices; check and complete documentation; gather feedback from the community and develop and adjust the project vision and long-term goals; advertise the project to gain new users and educate them in workshops. Infrastructure (test machines, self-hosted web-based services) has to be maintained as well although many hosted (and fully managed) solutions exists which can be freely used

for open-source projects (e.g., GitHub for code hosting and project management, Travis CI, travis-ci.org, for running automated tests) or can be self-hosted at relative small costs with the offerings from cloud service provider (e.g., Microsoft Azure, azure.microsoft.com, Google Cloud, cloud.google.com, Amazon Web Services, aws.amazon.com) on a pay-per-use basis.

4 Examples of Community-Driven Open-Source Software Development for the Simulation of Multi-physical Processes in Porous Media

In the following, the general concepts outlined above are complemented by insights into past and current development challenges and approaches as well as community activities based on the authors' work in the context of the projects DuMu^x, dumux.org, and OpenGeoSys, opengeosys.org. In particular, we will substantiate the rather general motivation for open-source development, the licensing issues and the process of community building described in Sect. 2, as well as exemplify the infrastructure components and development workflows introduced in Sect. 3. Despite this specificity, we hope that the points raised can be largely transferred to other software projects operating in similar ways.

4.1 DuMu^x—Evolving Development Paradigms in Long-Term Academic Software Projects

Over the course of years, developers gain experience, new tools become available, and development paradigms shift. When entering the world of open-source software, the scientist will be exposed to these changes. It is also important to recognize that different projects may be at different stages of a development process depending on the resources available to the project and the activity of its users. As an example for the evolution of development concepts of scientific computing software over time, the history of DuMu^x development is sketched in this section.

Initial development and first release The development of DuMu^x started in January 2007 at the Department of Hydromechanics and Modelling of Hydrosystems (LH2) at the University of Stuttgart. Before, the LH2 group was developing the simulator MUFTE-UG (Helmig et al. 1998) which was based on the PDE software framework UG (Bastian et al. 1997). Since the public development of and support for UG stopped for the time being in the early 2000s, the need arose for building on a different framework. The decision fell in favor of the C++ toolbox DUNE, the "Distributed and Unified Numerics Environment" (Bastian et al. 2008), motivated by the offered philosophy and functionality as well as by the fact that Peter Bastian, a main developer of UG and MUFTE-UG, belonged to the core developer team of DUNE and also stayed with his working group in the close proximity at the University of Stuttgart.

In March 2007, a Subversion repository was set up for DuMu^{x}to host and control the code development. From that point on, every new doctoral student and postdoc at the LH2 performed his/her modeling tasks by using and enhancing the program code in that repository. Up until today, all developers of DuMu^{x}have belonged to the LH2 working group. This naturally results in a continuous major goal of the project being to provide a tool that enables every developer to perform his research and possibly also teaching tasks. Motivated by the fact that the DUNE framework was open source and the wish to return something to the scientific community, the decision was made to publish DuMu^x under an open-source license. In July 2009, DuMu^x 1.0 was released under the GNU-GPL 2.0, see Sect. 2.2. While the DUNE framework is also issued under the GPL, it contains a so-called "runtime exception" allowing

that DUNE source files can be used as part of another software library or application without restriction. For DuMu^x, the choice for the standard version without the exception was made deliberately based on the motivation that all code using DuMu^{x}also has to be issued open source.

The 2.X release series DuMu^{x}1.0 consisted of a subset of the code stored in the Subversion repository because not everything in there was adequate for public release. Since selecting a subset of a private code base for public release proved to be rather impractical, the repository was split into a stable part `dumux` and a development part `dumux-devel` that was dependent on the stable part. Public read access to the repository of the stable part was granted. In `dumux-devel`, new capabilities of the software such as enhanced model concepts or new constitutive relations were and are to be added and used for the research work of the LH2. Once a new capability is considered to be stable and interesting enough for the scientific community, it can be moved to the stable part. In order to ensure the quality of added code and also all other changes to the stable part, each corresponding commit to the Subversion repository had to be reviewed internally by another DuMu^x developer.

Since the code design still exhibited several shortcomings concerning dependencies, generality and modularity, a major refactoring of the code base was performed in the following 1.5 years. As a result, DuMu^x 2.0 was released in February 2011 which also leads to the main reference Flemisch et al. (2011). After this release, interfaces should be kept stable for at least one release cycle in order to achieve more sustainability and security for the growing number of users and developers. Interface changes had to be discussed beforehand and in case of an actual change, the old interface had to be kept for the next release and its use should emit a compiler deprecation warning. By this policy, users of the software have been able to gradually adapt their own code after switching to a new release. The same guideline is followed in the DUNE framework. However, not all developers supported this policy change, arguing that it would slow down the development process. In March 2012, this leads to a fork of DuMu^{x}2.2 in form of the module `eWoms` (Lauser 2014). After a phase of DuMu^{x}changes being partially integrated into eWoms, the developments of the two modules are independent of each other for several years now, see also the paragraph on OPM below.

Following the tendency from centralized to distributed version control and particularly the DUNE framework, the DuMu^{x}Subversion repository was converted into Git repositories in September 2015. GitLab was employed as a version control management system, the Git repositories being publicly accessible at git.iws.uni-stuttgart.de. As outlined in Sect. 3.1, GitLab was chosen for being an open-source alternative and offering the possibility to run self-hosted on the working group's server infrastructure. Git, and in particular GitLab, greatly facilitated the joint development process. Changes to the code base are developed in branches and may be integrated into the mainline by an approved merge request. The approval has to be granted by somebody different than the author of the change, which formalizes and simplifies the aforementioned review process. The release process has been streamlined and since release 2.4 in October 2013, DuMu^x has been released biannually in spring and autumn every year, the last release of the 2.X series being 2.12 in late 2017. Since 2.7, every release tarball is uploaded to Zenodo, thereby receiving a DOI, as for example Fetzer et al. (2017).

Transition to DuMu^x*3.0* During the 2.X release series, several new features were added to the code base. Many of these additions fell in line with the main intention of DuMu^{x}to be a framework for the implementation of porous media model concepts and constitutive relations by actually providing such implementations. However, some more central additions had to be rather forced into the code base and proved to be inefficient, inconsistent with the original design ideas and/or increasingly difficult to maintain. For example, the spatial discretization

of the fully implicit models was rather hard-wired to the box scheme (Huber and Helmig 2000). While it was possible to add a cell-centered scheme with two-point flux approximation, the existing program flow, data structures and interfaces didn't fit well. Moreover, implementing a standard multi-point flux approximation scheme (Aavatsmark 2002) proved to be very tedious. The rather hard dependency on the box scheme also resulted in a suboptimal discretization for the free-flow models. While it was possible to add a corresponding stabilization, the implemented method wasn't robust enough for the envisioned range of applications.

Obstacles of different nature have been also encountered: For the coupling of porous medium and free flow, DuMu^{x}depended on the DUNE module `dune-multidomain` (Müthing 2015). Unfortunately, the development of this module was discontinued in 2014. Since `dune-multidomain` in turn depended on the modules `dune-pdelab` and `dune-multidomaingrid` and therefore on the DUNE core modules, it would have been necessary to adapt it to changes in these modules along with continuing the development and updating the dependencies of DuMu^{x}itself. The maintenance burden for this task was considered too high and it was decided to implement a model-coupling framework based on only the DUNE core modules, offering the additional advantage of being able to design that framework in line with the requirements posed by the targeted applications. Altogether, this leads to a new major release cycle that was initiated by branching off the main development line in November 2016. Due to the large amount of changes and their extent into all parts of the code base, the requirement of backward compatibility was dropped. Right after the release of DuMu^{x}2.12 in December 2017, the development branch was integrated back into the main line and an alpha release was published. It took another year before DuMu^{x}3.0 finally was released in December 2018. Including remedies for the shortcomings mentioned above, the list of improvements is rather long, dumux/CHANGELOG.md.

Quality assurance and reproducibility To assure the quality of the developed software, DuMu^x is accompanied by currently around 350 unit and system tests which are to be carried out by a BuildBot CI server at git.iws.uni-stuttgart.de/buildbot after each commit to the master branch. One guideline for developers is that every newly added feature has to be accompanied by a corresponding test. While adding tests should be a task that every researcher can perform as part of his daily routine, it is hard to develop a corresponding automated testing infrastructure as a mere by-product of scientific research. In the case of DuMu^x, this development was greatly facilitated by the project "Quality assurance in software frameworks on the example of DUNE/PDELab/DuMu^x," funded by the MWK Baden-Württemberg, the DFG Cluster of Excellence SimTech and the IWR at the University of Heidelberg.

In order to achieve reproducibility of the computational results, the project DuMu^x-Pub has been initiated in 2015. It provides a set of tools for the researcher to outsource the code that has been employed for a publication into a separate DUNE module, together with an install script to download and compile this code including all necessary dependencies. Since 2015, every journal publication at the LH2 as well as every bachelor, master and doctoral thesis has to be accompanied by such a module. All resulting modules are published at git.iws.uni-stuttgart.de/dumux-pub. First efforts have been undertaken to provide also a complete runtime environment in form of a Docker container https://git.iws.uni-stuttgart.de/dumux-pub/Koch2017a. These efforts will be streamlined in the future as part of the DFG-funded project "Sustainable infrastructure for the improved usability and archivability of research software on the example of the porous-media-simulator DuMu^x."

The Open Porous Media (OPM) initiative In 2009, the OPM initiative was born with the principal objective "to develop a simulation suite that is capable of modeling industrially

and scientifically relevant flow and transport processes in porous media and bridge the gap between the different application areas of porous media modeling" (Lie et al. 2009). With the SINTEF Simulation Group (Oslo), the IRIS Reservoir Group (Bergen), the IWR (Heidelberg), the LH2 (Stuttgart) and the Center of Integrated Petroleum Research (Bergen), five research groups initially joined OPM. Actual work on OPM started by means of two projects between 2010 and 2013: "Simulation of EOR in Clastic Reservoirs" funded by Statoil and Total as well as "A numerical CO_2 laboratory" funded by CLIMIT, www.climit.no, and Statoil. During these years, SINTEF and IRIS constituted themselves as the main contributing groups to the simulator part of the emerging OPM code base, which they still can be considered today. Currently, the main focus of OPM is on reservoir engineering with a particular emphasis on being able to compete with proprietary industry-standard tools. As such, the black-oil simulator `Flow` is a main product of OPM. Meanwhile, several additional funding and contributing partners joined the initiative and its spectrum has been enhanced by upscaling and visualization modules.

DuMu^{x}initially contributed to OPM in form of the eWoms module mentioned above, as the developer of eWoms joined the OPM development team. In particular, the module `opm-material` originated from the DuMu^{x}material framework and `eWoms` meanwhile constitutes the basis of the module `opm-simulators`. Parallel to the development of DuMu^x, the OPM modules have undergone significant changes since then. Up to now, the DuMu^{x}code base did not profit directly from the associated improvements by integrating (some of) these changes. Nevertheless, DuMu^{x}benefits from OPM in several aspects, directly by the fact that DuMu^{x}has a suggested dependency on the OPM module `opm-grid` which enables every DuMu^{x}user to employ easily corner-point grids that are standard in the petroleum industry. Indirectly, DuMu^{x}and OPM profit from each other in the form of joint activities and a lively exchange of ideas and details driven by the facts that both development teams use DUNE components and target similar applications. This is expected to be intensified by means of the project "InSPiRE," iris.no. As of today, DuMu^{x}is considered to be "related" to the OPM initiative, opm-project.org.

Community building By going open source, external users of DuMu^{x}have been welcome ever since the initial release in 2009. Between 2009 and 2018, tarballs of DuMu^{x}releases could be obtained by submitting name, institution and email address via the website. Over the years, around 1200 unique submissions from apparently "serious" email addresses could be counted. The number of Subversion checkouts and Git clones has not been recorded so far. While these are only measures for interest rather than usage, we scan frequently the scientific publications that mention DuMu^x. So far, about 25 articles and 6 doctoral theses from around 10 research groups have been identified that cite and actually use DuMu^{x}for obtaining computational results without being co-authored by a current or former LH2 member.

In order to get in contact with the users, a first DuMu^{x}user meeting was held in Stuttgart in June 2015. Twenty-six participants were counted; ten of them were external users. For attracting new users, a first DuMu^{x}course was given in October 2017, followed by a second one in July 2018. This proved to be a great measure, indicated by increased traffic on the DuMu^{x}mailing list, mainly due to former course participants. While there had been around 5 posts per month prior to the first course (apart from automated notifications by the former issue tracker), this has increased to more than 35 per month during the second half of 2018.

With GitLab, all technical possibilities are available for users to upload contributions to the code base in form of merge requests as well as for developers to review, discuss, improve and possibly integrate them. However, contributions from outside the LH2 are very scarce until now. Current notable exceptions are members from the neighboring Institute of

Applied Analysis and Numerical Simulation (IANS) at the University of Stuttgart as well as researchers from the Federal Waterways Engineering and Research Institute (BAW) in Karlsruhe. With the apparent increase in interest and actual usage over the last years, we hope to increase as well the number of external contributors and, ultimately, developers in the future.

4.2 OpenGeoSys—Flexible Software Engineering Workflows for Quality Assurance

Development of OpenGeoSys-6 started in 2011 and was open-sourced 2012 under the 4-clause BSD license. It took 3 years of developing basic data structures and general software architecture based on the experience from OpenGeoSys-5 (Kolditz et al. 2012) until 2015 when version 6.0.1 was released with the first implemented physical process. Major milestones were achieved in 2016 by implementing a parallel computing framework based on PETSc and since 2017 the extensive work on the general software architecture pays off with lots of process implementations following. We typically do a release once a year but consider every change which was merged to our main code line as "production-ready." Therefore, we established automatic procedures to continuously deliver the up-to-date and fully tested application to our users on a daily basis (which is described in the following paragraphs). Releases are available at Zenodo as well (Fischer et al. 2018; Naumov et al. 2019).

Community building The project is available on GitHub and open for external contributions. A large number of external contributors have been working with OpenGeoSys-5 for many years. A growing feature list, an increased number of reference solutions and an improved periphery make a transition to the new version OpenGeoSys-6 increasingly attractive. Nevertheless, such a transition can be challenging for external user groups (see Continuity vs. Change in Sect. 5). In the past year we had submissions from the Federal Institute for Geosciences and Natural Resources (BGR, Germany), National Institute of Advanced Industrial Science and Technology (AIST, Japan) and Technische Universität Bergakademie Freiberg but we expect an increase in contributions as the code base has now reached some maturity level. The OpenGeoSys community meeting has been taking place once a year since 2010 with an audience of 30–50 people. To provide a sustainable communication channel to users we use mailing lists but also started exploring a Discourse forum, discourse.opengeosys.org, because it features file attachments, optional private groups, customizable notifications, better cross-linking and search functionality. We also established a monthly UFZ internal user meeting and publish a tutorial book series (Lehmann et al. 2018) on specific scientific applications of OpenGeoSys which are intended as training material. A benchmark book series (Kolditz et al. 2018a) regularly gathers specific examples of code verification and benchmarking initiatives. Aside from pure documentation and quality assurance, the individual chapters in these books serve the collaboration with users outside of the core developer group and across institutions. We try to attract new users by providing ready-to-use binaries in combination with an extensive cross-linked documentation showcasing selected applications and explaining typical workflows and model setup (www.opengeosys.org/docs/).

Specific challenges A flexible development approach and the need to provide a variety of interfaces and integrations to other numerical codes results in an elevated testing demand for the developed software. Testing every possible configuration is not feasible but the most common configurations are thoroughly tested and the implementation of a scalable testing approach is pursued in OpenGeoSys. Handling this complexity—ensuring future usability and limiting the possibility of feature loss during the course of the development—requires

strict quality assurance (QA) measures by using automated tests and quality checks especially in the context of a distributed developer team diverse in professional backgrounds. The QA process itself as well as its required hardware and software needs to be strictly defined and should be transparent to the scientists and stakeholders involved. Furthermore, changes to the process should be easily implementable; required hardware and software should be easily deployable even by non-experts as supportive staff is limited, particularly in the scientific context.

Approach To tackle these challenges, OpenGeoSys makes use of methods and technologies best described by the term DevOps (Loukides 2012; Kim et al. 2016)—a clipped compound of software development and information technology operations. DevOps as described in de Bayser et al. (2015) is "a software development method that extends the agile philosophy to rapidly produce software products and services and to improve operations performance and quality assurance." The authors propose the usage of related technologies in scientific software projects such as (i) infrastructure as code (IaC) to automate the setup of unified version-controlled development environments with no manual configuration necessary, (ii) continuous delivery (CD) pipelines with automated software testing and as a consequence to offer (iii) software-as-a-service (SaaS) where—similar to web applications—software is continuously updated and delivered to the user after successful verification and validation to shorten iteration cycles. All of these technologies have in common that they use domain specific languages (DSLs) and corresponding tools to specify and implement a certain aspect of the development process.

Quality assurance and continuous integration OpenGeoSys-6 is developed as an open-source project by using the GitHub platform for code hosting, code review as well as bug and issue tracking, see github.com/ufz/ogs. Due to storage limitations on GitHub OpenGeoSys uses a sponsored Artifactory (jfrog.com/artifactory) instance at ogs.jfrog.io for storing large Git LFS files for, e.g., benchmark input and reference data. A Jenkins continuous integration (CI) server at jenkins.opengeosys.org builds and tests the software on every proposed change (pull request) on supported platforms (Windows, Linux, Mac Desktop systems and Linux HPC cluster systems). The current test setup includes code style checks, reports on compiler warnings or possible memory leaks detected by static code analyzers. Code behavior is tested with unit tests and end-to-end tests in the form of benchmark runs and the comparison of their result data files in respect to analytical and reference solutions. Comprehensive documentation, generated automatically from code comments as well as handwritten documents on topics such as tool and benchmark explanations and developer instructions, is deployed automatically to opengeosys.org/docs as part of the CI process. See Fig. 2 for an overview of the development workflow. The QA process is automated and defined via the Jenkins Pipeline DSL, jenkins.io/doc/book/pipeline/syntax, allowing pull request authors to even modify the QA process, e.g., to add new test configuration or to temporarily disable specific parts of the testing setup to shorten feedback cycles from the QA system to the developer. Developers can also automatically test their personal development repositories with the help of QA system if required. The mandatory hardware for the QA process is hosted locally at the Helmholtz Centre for Environmental Research — UFZ as well as hosted at a cloud platform service provider and is administrated with the help of the Ansible configuration management DSL and tools, docs.ansible.com/ansible/playbooks.html. The configuration DSL files are available as a public repository at gitlab.opengeosys.org/bilke/ogs-ansible. We provide Linux container based test environments for multiple software configurations (i.e., for different parallelization schemes or numerical solver libraries) based on Dockerfile (docs.docker.com/engine/reference/builder) definitions which are available as a public

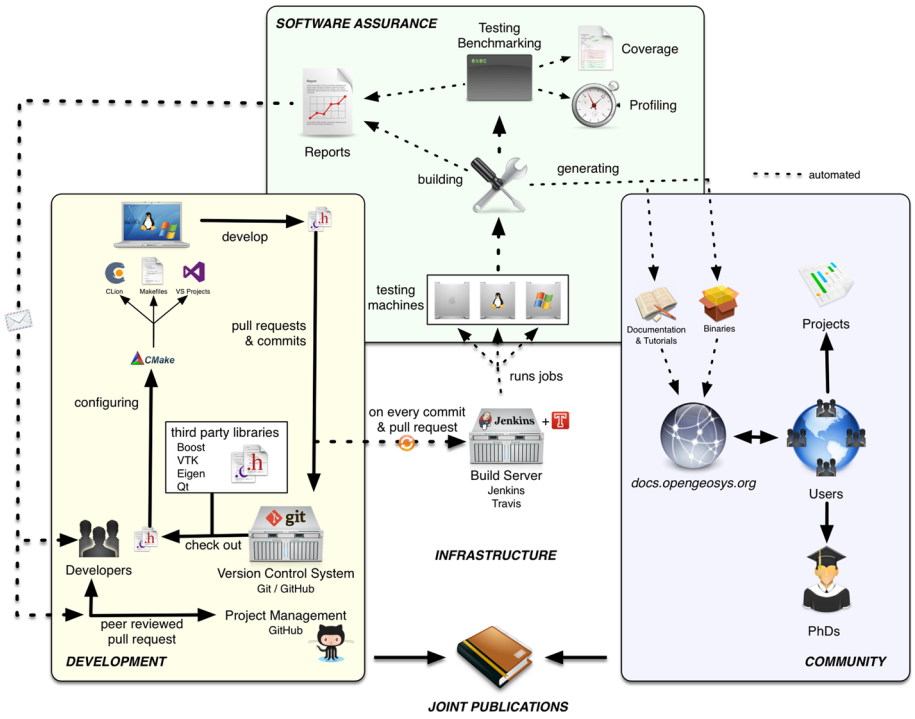

Fig. 2 OpenGeoSys software engineering workflow overview

repository at github.com/ufz/dockerfiles. Ready-to-use pre-built container images can be downloaded from Docker Hub at hub.docker.com/u/ogs6.

Outlook We are exploring Python interfaces for easy extension of the software (currently implemented for defining solution-dependent boundary conditions and source terms) and containerization technologies such as Docker and Singularity for easy distribution of binaries especially in the HPC context (Kurtzer et al. 2017). A container generator (github.com/ufz/ogs-container-maker) based on McMillan (2018) will provide an easy way for users to create specific configurations (e.g., by specifying the Git repository and branch, configuration options, the compiler toolchain, etc.) of OpenGeoSys in an automated fashion (by using our CI system) without any setup of a development environment. Generated containers are portable and can be easily archived thus helping in reproducing scientific results.

5 Lessons Learned and Conclusions

This concluding section is split into two parts. In Sect. 5.1, we reflect on the lessons learned in coordinating two medium-sized open-source projects over several years. Some more general conclusions are provided in Sect. 5.2.

5.1 Lessons Learned

Following our experiences gained in DuMu^{x}and OpenGeoSys as described in Sect. 4, we span six edges representing potential areas of conflicts. Along each such edge, optima have to be identified specifically for a project that might possibly change over time.

Rapid expansion vs. incremental sustainability Research progresses at an increasingly fast pace. Associated with a desired progress usually comes the demand of extending the capabilities of the corresponding software. Such a rapid extension may be tempting for the developing group since more capabilities may easily be equated with a better product. However, a rapid expansion strategy usually is opposed to a potential desire for a sustainable development and maintainable code. Possibly, code components are integrated that are only needed and understood by an individual person. Once that person leaves, the added feature remains unused and constitutes a maintenance burden. Therefore, it should be weighed which new features become integrated into the main code base. Git branches and outsourcing into separate modules offer short-term and long-term alternatives where new additions can mature and possibly still be integrated at a later stage. For example, the DUNE developer community is rather reserved about adding extensions to their core modules. Corresponding additions have to be discussed at length beforehand and agreed upon, sometimes requiring a formal vote. Extensions falling out of the clearly defined scope of the core modules are more likely to be outsourced to possibly new modules outside the core. Since the scope of DuMu^{x}is rather vague in the sense of "a simulator for a large variety of problems related to porous media," extensions are more easily justified, of course after having matured in a Git branch or another module. For OpenGeoSys every part of the code considered as general software infrastructure is developed in a sustainable way, i.e., it is thoroughly discussed, reviewed, iterated and tested before it becomes part of the code main line. For more specialized parts, e.g., a concrete process implementation, a more rapid expansion is allowed as long as formal requirements are met and sufficient tests for the added functionality are provided.

Continuity vs. change While a continuous development creates reliability of the project, it also bears the danger of stagnation. Improvement requires change, but too disruptive changes can alienate and annoy users and fellow developers. Every project should have a clear understanding of what are the code parts and interfaces that may be changed and what are the ones that should better be not touched. Interface changes should be carefully planned and discussed beforehand as well as, if feasible, performed in a backward compatible way. As described in Sect. 4.1, violations of this guideline ultimately led to a fork and corresponding diverging developments. The transition from OGS-5 to OGS-6 has likewise been a stark change which the developer team tried to dampen by continuing some level of support for OGS-5. The edge can also be interpreted in terms of personnel. In academia, change is the default as the usual developer is a doctoral student or postdoc who contributes to the project only for a limited time. A change of developers comes along with the chance of bringing in new ideas and expertise. However, for a sustainable project, it is of crucial importance to complement this with developers in long-term positions. Unfortunately, this is still difficult in academia today. In case of DuMu^x, the main development efforts are undertaken by doctoral students and postdocs, while one permanent position exists with the major task of coordinating these efforts. OpenGeoSys is developed by four core developers which manage the general direction, give support and implement and maintain core software architecture

with well-defined extension points used by domain scientists for their application-oriented process implementations.

Modularization vs. integration As a code base grows, the question of splitting it up arises naturally, driven by the expectation that the resulting parts are more focused, easier to maintain and that users who are only interested in a specific functionality only have to download and handle the corresponding parts. Similarly, a larger new feature might have been implemented in a separate module and appears to be mature and interesting enough to be integrated in the main module. Should it actually be integrated or stay separate? It is very difficult to provide a general answer. While the positive expectations resulting from having a set of smaller modules as opposed to a single integrated one may come true, modularization also comes with potential disadvantages. If it's overdone, a user will have a hard time understanding which particular modules he needs and how they interact. Developers will only look at the modules in which they are interested and rarely used modules won't receive much attention. Modularization also poses challenges to regression testing. While one usually would like to test if a change breaks code before integrating it into the main line, this task becomes more difficult to be performed automatically for downstream modules. As already indicated in the discussion of the first edge, the DUNE philosophy is embracing modularization which is opposed to the claim of DuMu^x of being applicable to a large range of porous media-related problems and OpenGeoSys is also developed in a monolithic repository.

Do-it-yourself vs. off-the-shelf A very frequently encountered standard argument is "Don't try to reinvent the wheel." This certainly holds true if a choice is about implementing a data structure/feature, etc., from scratch versus a solution that is well defined, mature and available from a trusted, sustainable source. One should not try to reimplement a `std::vector` from the C++ standard library. However, an existing solution should be carefully evaluated if it doesn't fulfill these measures, for example being a new, highly specialized tool from an individual scientist's research work. While it might be perfectly fine to depend on the corresponding piece of code for another individual workpiece, adding such a dependency to a code base that is supposed to undergo a continuous and sustainable development is dangerous. It might soon become a maintenance burden once the upstream development stops or undergoes disruptive changes. In such a case, it can be a better long-term solution to develop the desired component oneself with the advantage of having full control over this development. As described in Sect. 4.1, DuMu^x depended on the module dune-multidomain for solving coupled problems. With the discontinued development of that module, it was necessary to develop the corresponding capabilities inside DuMu^x itself. This came with the positive effect of being able to fit these new capabilities to the existing interfaces as well as the freedom to design them to be in line with the code basis. The OpenGeoSys development team is experienced in numerics and process implementation which therefore sets the main development focus. For supportive code such as linear algebra, parallelization schemes, testing frameworks or visualization interfaces, only well-maintained and stable third-party libraries are carefully chosen and employed. This has been facilitated in the past years due to the existence of high-quality open-source libraries developed at other institutions.

Isolation vs. communication While development of new features or improvements in isolation may lead to faster results, it is dangerous to integrate such a result unreflected into the code base. Formalized reviews help, but they don't ensure that the integrated code is free of bugs and neither that it is adequately implemented for the whole range of desired use cases. Communication is key to discuss intended changes beforehand. While a discussion possibly slows down the process, the result can be expected to be of improved quality as

well as to be better understood and supported by the development team. When a bigger change for DuMu^{x}is planned, it is required to open a GitLab issue or merge request to initiate a corresponding discussion. In addition, monthly developer meetings are organized for accelerating the decision making. Every change in the OpenGeoSys code base is required to have a corresponding pull request on GitHub and will be reviewed by at least two core developers. Larger monolithic changes are often requested to be modularized and split into multiple smaller pull requests for easier review and to foster code abstraction/generalization.

Distributed vs. centralized This is not about version control, where the answer is settled, but about the composition of the development team itself. It may be inevitable and is very often desirable that the development is distributed over several places. It comes with the obvious advantage that the expertise of different individuals and working groups can be combined successfully. Luckily, distributed development is technically feasible and greatly facilitated by many existing tools. Nevertheless, the value of personal meetings can hardly be overestimated. Bringing all developers together in one room provides the possibility to bring about important decisions within a few hours or days as opposed to possibly year-long discussions via the issue tracker. Obviously, this might not be feasible for projects that really span around the globe, in which case meetings in virtual rooms can be a compromise. For example, the development of the DUNE core modules is distributed across Europe. Larger decisions on the direction of the project are undertaken at personal developer meetings that take place approximately once a year. The rather centralized development teams of DuMu^{x}and OpenGeoSys allow for more frequent monthly meetings, as already mentioned above.

5.2 Conclusions

Numerical simulations have become an established means of performing collaborative research in the diverse fields addressing porous media relying on open-source software projects for scientific computing as research infrastructures.

The developments described in this article affect not only the scientific community but also industry and government agencies both indirectly and directly, as evidenced by the emergence of open-source business models (Tapscott and Williams 2006; Will Schroeder 2010), an aspect not addressed in this article. With increased public awareness and the advent of increasingly controversial, high-consequence or large-scale technologies (e.g., induced seismicity from subsurface stimulation, fate of contaminants and radionuclides in the environment, novel biotechnologies), stakeholders, including policy makers and the public, often demand rapid development of specific solutions or answers in discussions on civil protection against natural disasters. Often, scientists with expertise in porous media are among the addressees of such questions. Transparency, independence and accountability are key to public acceptance and informed decision making, and there is good reason to believe that open-source frameworks can contribute in this regard. Key requirements are sufficient access to infrastructure, established quality assurance procedures (which includes benchmarking using open data) as well as an active skilled user and developer community. In view of these wider benefits, it is important to recognize that open-source software is not “for free” but relies on fundamental development, plain maintenance and adequate development infrastructures. This maintenance and professional improvement of the technical environment and the involved highly qualified staff require both enthusiasm and continuous support.

In this context, it is remarkable that the development processes in the two software projects coordinated by the authors are in several aspects closer to the processes encountered in companies of a correspondingly small size than to those found in large community-based

open-source projects. This originates from the fact that the development teams of both projects are small and centralized and not a lot of contributions have been made so far from persons outside of these teams. While such contributions have been declared to be welcome and the necessary workflows have been established, it still remains to be seen if these workflows will be adopted by the user communities and actual co-developers will emerge from there.

Good scientific practice entails the reproducibility of research and as a consequence also the accessibility and documentation of any numerical code and related data which has been employed in the course of a project. Code quality assurance measures, such as automated testing, code review and documentation, are essential for establishing confidence into models and software alike, in particular in dynamic communities where experts leave frequently. Editors of scientific journals and reviewers of scientific papers can contribute to this process as well by encouraging the use of open software, open data, and generally, keeping an eye on reproducibility. This will not always be immediately possible but can certainly become a trend and should be an objective.

We hope that by giving insights into relevance and procedures of open-source software development we have motivated some of the readers to engage in the open-source community, to share their knowledge and help increase not only our understanding of how porous media behave but also improve the computational tools required in this fascinating quest.

Acknowledgements We would like to express our thanks to the community of developers and users of the described software, in particular to those involved in the long-term projects OpenGeoSys (OGS) and DuMu^x, in which the authors are involved. We thank the Helmholtz Centre for Environmental Research — UFZ for long-term funding and continuous support of the OpenGeoSys initiative (Helmholtz future projects Earth System Modeling (ESM) and Digital Earth). OGS has been supported by various projects funded by Federal Ministries (BMBF, BMWi) as well as the German Research Foundation (DFG). We further thank the Federal Institute for Geosciences and Natural Resources (BGR) for funding and the DECOVALEX initiative for providing an open and productive research environment for the development and validation of coupled models. The OpenGeoSys community thanks Microsoft for sponsoring a part of the QA infrastructure of OGS with their sponsorships for nonprofit organizations. The sustainable development of DuMu^{x}profited and profits largely from the projects "Quality assurance in software frameworks on the example of DUNE/PDELab/DuMu^x" funded by the MWK Baden-Württemberg, the DFG Cluster of Excellence SimTech and the IWR at the University of Heidelberg as well as "Sustainable infrastructure for the improved usability and archivability of research software on the example of the porous-media-simulator DuMu^x" funded by the German Research Foundation DFG. Finally, we thank the representatives and members of the International Society for Porous Media, InterPore, for the establishment of a vibrant community and discussion platform dedicated to "bridging the gap" between different research disciplines and entities.

References

Aavatsmark, I.: An introduction to multipoint flux approximations for quadrilateral grids. Comput. Geosci. **6**, 405–432 (2002). https://doi.org/10.1023/A:1021291114475

Ambrosi, D., Ateshian, G.A., Arruda, E.M., Cowin, S.C., Dumais, J., Goriely, A., Holzapfel, G.A., Humphrey, J.D., Kemkemer, R., Kuhl, E., Olberding, J.E., Taber, L.A., Garikipati, K.: Perspectives on biological growth and remodeling. J. Mech. Phys. Solids **59**(4), 863–883 (2011). https://doi.org/10.1016/j.jmps.2010.12.011

Bangerth, W., Hartmann, R., Kanschat, G.: deal. ii—a general-purpose object-oriented finite element library. CM Trans. Math. Softw. (TOMS) **33**(4), 24 (2007)

Bastian, P., Birken, K., Johannsen, K., Lang, S., Neuß, N., Rentz-Reichert, H., Wieners, C.: Ug—a flexible software toolbox for solving partial differential equations. Comput. Vis. Sci. **1**(1), 27–40 (1997). https://doi.org/10.1007/s007910050003

Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Kornhuber, R., Ohlberger, M., Sander, O.: A generic grid interface for adaptive and parallel scientific computing. Part II Implement. tests DUNE **82**(2–3), 121–138 (2008)

Baxendale, D., Rasmussen, A., Rustad, A.B., Skille, T., Sandve, T.H.: Open Porous Media Flow Documentation Manual. (2018). https://opm-project.org/wp-content/uploads/2018/11/OPM-Flow-Documentation-2018-10-Rev-1.pdf

de Bayser, M., Azevedo, L.G., Cerqueira, R.: Researchops: The case for devops in scientific applications. In: PROCEEDINGS, IFIP/IEEE International Symposium on Integrated Network Management, pp. 1398–1404 (2015)

Bazant, Z.P., Gattu, M., Vorel, J.: Work conjugacy error in commercial finite-element codes: its magnitude and how to compensate for it. Proc. R. Soc. A Math. Phys. Eng. Sci. **468**(2146), 3047–3058 (2012). https://doi.org/10.1098/rspa.2012.0167

Boehm, B.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)

Bradley, C., Bowery, A., Britten, R., Budelmann, V., Camara, O., Christie, R., Cookson, A., Frangi, A.F., Gamage, T.B., Heidlauf, T., et al.: OpenCMISS: a multi-physics & multi-scale computational infrastructure for the VPH/Physiome project. Prog. Biophys. Mol. Biol. **107**(1), 32–47 (2011)

BSD (2018) Bsd licenses — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=BSD_licenses&oldid=873127188, [Online; Accessed 12 Dec 2018]

Carrel, M., Morales, V.L., Beltran, M.A., Derlon, N., Kaufmann, R., Morgenroth, E., Holzner, M.: Biofilms in 3D porous media: Delineating the influence of the pore network geometry, flow and mass transfer on biofilm development. Water Res. **134**, 280–291 (2018). https://doi.org/10.1016/j.watres.2018.01.059

Class, H., Ebigbo, A., Helmig, R., Dahle, H.K., Nordbotten, J.M., Celia, M.A., Audigane, P., Darcis, M., Ennis-King, J., Fan, Y., Flemisch, B., Gasda, S.E., Jin, M., Krug, S., Labregere, D., Naderi Beni, A., Pawar, R.J., Sbai, A., Thomas, S.G., Trenty, L., Wei, L.: A benchmark study on problems related to CO2 storage in geologic formations. Comput. Geosci. **13**(4), 409–434 (2009). https://doi.org/10.1007/s10596-009-9146-x

Coussy, O.: Mechanics and Physics of Porous Solids. Wiley, Chichester (2010). https://doi.org/10.1002/9780470710388

Forschungsgemeinschaft, Deutsche: Proposals for Safeguarding Good Scientific Practice. Wiley-VCH, Weinheim (1998)

Diersch, H.: Modellierung und numerische Simulation gehydrodynamischer Transportprozesse. Akademie der Wissenschaften der DDR, Habilitationsschrift, Berlin (1984)

Diersch, H.J.: FEFLOW: Finite element modeling of flow, mass and heat transport in porous and fractured media. Springer, (2014). https://doi.org/10.1007/978-3-642-38739-5, cited By 156

Ehlers, W.: Foundations of multiphasic and porous materials. In: Ehlers, W., Bluhm, J. (eds.) Porous Media: Theory, Experiments and Numerical Applications, pp. 4–86. Springer, Berlin (2002)

Ehlers, W., Häberle, K.: Interfacial mass transfer during gas–liquid phase change in deformable porous media with heat transfer. Transp. Porous Media **114**(2), 525–556 (2016). https://doi.org/10.1007/s11242-016-0674-2

European Commission (2018) Commission recommendation (eu) 2018/790 of 25 april 2018 on access to and preservation of scientific information c/2018/2375. http://data.europa.eu/eli/reco/2018/790/oj

Fetzer, T., Becker, B., Flemisch, B., Gläser, D., Heck, K., Koch, T., Schneider, M., Scholz, S., Weishaupt, K.: Dumux 2.12.0. (2017). https://doi.org/10.5281/zenodo.1115500

Fischer, T., Naumov, D.Y., Bilke, L., Rink, K., Lehmann, C., Watanabe, N., Wang, W., Huang, Y., Miao, X., Walther, M., Zheng, T., Parisio, F., Helbig, C., English, M.: ufz/ogs: 6.1.0. (2018). https://doi.org/10.5281/zenodo.1145843

Flemisch, B., Darcis, M., Erbertseder, K., Faigle, B., Lauser, A., Mosthaf, K., Müthing, S., Nuske, P., Tatomir, A., Wolff, M., Helmig, R.: DuMuX: DUNE for Multi-{phase, component, scale, physics, . . .} flow and transport in porous media. Adv. Water Resour. **34**(9), 1102–1112 (2011)

Flemisch, B., Berre, I., Boon, W., Fumagalli, A., Schwenck, N., Scotti, A., Stefansson, I., Tatomir, A.: Benchmarks for single-phase flow in fractured porous media. Adv. Water Resour. **111**, 239–258 (2017). https://doi.org/10.1016/j.advwatres.2017.10.036

Fomel, S., Claerbout, J.F.: Reproducible research. Comput. Sci. Eng. **11**(1), 5–7 (2009)

Fuggetta, A.: Open source software—an evaluation. J. Syst. Softw. **66**, 77–90 (2003)

Gaston, D., Newman, C., Hansen, G., Lebrun-Grandié, D.: Moose: a parallel computational framework for coupled systems of nonlinear equations. Nucl. Eng. Des. **239**, 1768–1778 (2009)

González-Barahona, J.M., Pascual, J.S., Robles, G.: Introduction to Free Software. Free Technology Academy, Amsterdam (2013)

GPL (1991) GNU General Public License, version 2. https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html

Gray, W., Hassanizadeh, S.: Macroscale continuum mechanics for multiphase porous-media flow including phases, interfaces, common lines and common points. Adv. Water Resour. **21**(4), 261–281 (1998)

Görke, U.J., Günther, H., Nagel, T., Wimmer, M.A.: A large strain material model for soft tissues with functionally graded properties. J. Biomech. Eng. **132**(7), 074,502 (2010). https://doi.org/10.1115/1.4001312
Helmig, R., et al.: Multiphase Flow and Transport Processes in the Subsurface: A Contribution to the Modeling of Hydrosystems. Springer, Berlin (1997)
Helmig, R., Class, H., Huber, R., Sheta, H., Ewing, R., Hinkelmann, R., Jakobs, H., Bastian, P.: Architecture of the modular program system MUFTE-UG for simulating multiphase flow and transport processes in heterogeneous porous media. Math. Geol. **2**(123–131), 64 (1998)
Huber, R., Helmig, R.: Node-centered finite volume discretizations for the numerical simulation of multiphase flow in heterogeneous porous media. Comput. Geosci. **4**(2), 141–164 (2000). https://doi.org/10.1023/A:1011559916309
Hutter, K., Jöhnk, K.: Continuum Methods of Physical Modeling: Continuum Mechanics, Dimensional Analysis, Turbulence. Springer, Berlin (2004)
Huyghe, J., Janssen, J.: Quadriphasic mechanics of swelling incompressible porous media. Int. J. Eng. Sci. **35**(8), 793–802 (1997)
Islam, A.W., Sepehrnoori, K.: A review on SPE's comparative solution projects (CSPs). J. Pet. Sci. Res. **2**(4), 167–180 (2013)
Ji, W., Waas, A.M., Bazant, Z.P.: On the importance of work-conjugacy and objective stress rates in finite deformation incremental finite element analysis. J. Appl. Mech. **80**(4), 041,024 (2013). https://doi.org/10.1115/1.4007828
Keilegavlen, E., Fumagalli, A., Berge, R., Stefansson, I., Berre, I.: PorePy: An Open-Source Simulation Tool for Flow and Transport in Deformable Fractured Rocks. (2017). arXivorg http://arxiv.org/abs/1712.00460
Kelly, D.F.: A software chasm: software engineering and scientific computing. IEEE Softw. **24**(6), 119–120 (2007). https://doi.org/10.1109/MS.2007.155
Kelly, D.F., Sanders, R.: Assessing the quality of scientific software. In: PROCEEDINGS, First International Workshop on Software Engineering in Computational Science and Engineering (2008)
Kempf, D., Koch, T.: System testing in scientific numerical software frameworks using the example of dune. Arch. Numer. Softw. **5**(1), 151–168 (2017). https://doi.org/10.11588/ans.2017.1.27447
Kim, G., Humble, J., Debois, P., Willis, J.: The DevOps Handbook. IT Revolution Press, Portland (2016)
Kitzes, J., Turek, D., Deniz, F.: The Practice of Reproducible Research: Case Studies and Lessons from the Data-intensive Sciences. Univ of California Press, Berkeley (2017)
Kolditz, O.: Computational Methods in Environmental Fluid Mechanics. Springer, Berlin (2002). https://doi.org/10.1007/978-3-662-04761-3
Kolditz, O., Görke, U.J., Shao, H., Wang, W.: Thermo-Hydro-Mechanical-Chemical Processes in Fractured Porous Media: Benchmarks and Examples, vol. 86. Springer, Berlin (2012)
Kolditz, O., Nagel, T., Shao, H., Wang, W., Bauer, S. (eds.): Thermo-Hydro-Mechanical-Chemical Processes in Fractured Porous Media: Modelling and Benchmarking. Terrestrial Environmental Sciences, Springer International Publishing, Cham, (2018a). https://doi.org/10.1007/978-3-319-68225-9
Kolditz, O., Nagel, T., Shao, H., Wang, W., Bauer, S. (eds.): Thermo-Hydro-Mechanical-Chemical Processes in Fractured Porous Media: Modelling and Benchmarking: From Benchmarking to Tutoring. Springer (2018b). https://doi.org/10.1007/978-3-319-68225-9
Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: scientific containers for mobility of compute. PLOS ONE **12**(5), e0177,459 (2017). https://doi.org/10.1371/journal.pone.0177459
Lauser, A.: Theory and numerical applications of compositional multi-phase flow in porous media. Ph.D. thesis, University of Stuttgart, (2014). https://doi.org/10.18419/opus-516
Lehmann, C., Kolditz, O., Nagel, T.: Models of thermochemical heat storage. Comput. Model. Energy Syst. (2018). https://doi.org/10.1007/978-3-319-71523-0
Lichtner, P.C., Hammond, G.E., Lu, C., Karra, S., Bisht, G., Andre, B., Mills, R., Kumar, J.: Pflotran user manual: A massively parallel reactive flow and transport model for describing surface and subsurface processes. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States); Sandia.. (2015)
Lie, K., Krogstad, S., Ligaarden, I.S., Natvig, J.R., Nilsen, H.M., Skaflestad, B.: Open-source matlab implementation of consistent discretisations on complex grids. Comput. Geosci. **16**(2), 297–322 (2012). https://doi.org/10.1007/s10596-011-9244-4
Lie, K.A.: An Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox (MRST). Cambridge University Press, Cambridge (2019)
Lie, K.A., Bastian, P., Dahle, H.K., Flemisch, B., Flornes, K., Rasmussen, A., Rustad, A.B.: OPM—open porous media. Unpublished (2009)
Logg, A., Mardal, K.A., Wells, G.: Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book, vol. 84. Springer, Berlin (2012)
Loukides, M.: What is DevOps? O'Reilly Media (2012)

Maas, S.A., Ellis, B.J., Ateshian, G.A., Weiss, J.A.: Febio: finite elements for biomechanics. J. Biomech. Eng. **134**(1), 011,005 (2012). https://doi.org/10.1115/1.4005694

Maxwell, R., Condon, L., Kollet, S.: A high-resolution simulation of groundwater and surface water over most of the continental us with the integrated hydrologic model parflow v3. Geosci. Model Develop. **8**(3), 923 (2015)

McDonald, M., Harbaugh, A.: The history of MODFLOW. Ground Water **41**(2), 280–283 (2003)

McMillan, S.: Making Container Easier with HPC Container Maker. In: In HPCSYSPROS18: HPC System Professionals Workshop, Dallas, TX, (2018). https://github.com/HPCSYSPROS/Workshop18/tree/master/Making_Container_Easier_with_HPC_Container_Maker

Morin, A., Urban, J., Sliz, P.: A quick guide to software licensing for the scientist-programmer. PLoS Comput. Biol. **8**(7), e1002,598 (2012)

Müthing, S.: A flexible framework for multi physics and multi domain PDE simulations. Ph.D. thesis, University of Stuttgart, (2015). https://doi.org/10.18419/opus-3620

Nagel, T., Beckert, S., Lehmann, C., Gläser, R., Kolditz, O.: Multi-physical continuum models of thermochemical heat storage and transformation in porous media and powder beds—a review. Appl. Energy **178**, 323–345 (2016). https://doi.org/10.1016/j.apenergy.2016.06.051

Naumov, D.Y., Fischer, T., Bilke, L., Rink, K., Lehmann, C., Watanabe, N., Wang, W., Huang, Y., Lu, R., Chen, C., Bathmann, J., Miao, X., Yoshioka, K., Shao, H., Walther, M., Zheng, T., Parisio, F., Thiele, J., Grunwald, N., Helbig, C., Buchwald, J., Nagel, T.: ufz/ogs: 6.2.0. (2019). https://doi.org/10.5281/zenodo.2600045

Oberkampf, W.L., Trucano, T.G., Hirsch, C.: Verification, validation, and predictive capability in computational engineering and physics. Appl. Mech. Rev. **57**(5), 345–384 (2004). https://doi.org/10.1115/1.1767847

Open Source Initiative: What is "free software" and is it the same as "open source"?. (2018). https://opensource.org/faq#free-software, Accessed 11 Dec 2018

Prud'Homme, C., Chabannes, V., Doyeux, V., Ismail, M., Samake, A., Pena, G.: Feel++: A computational framework for galerkin methods and advanced numerical methods. In: ESAIM: Proceedings, EDP Sciences, vol. 38, pp. 429–455 (2012)

Puder, A.: Ubiquitous computing environments through open systems. In: Patel, D., Choudhury, I., Patel, S., de Cesare, S. (eds.) OOIS'2000, 6th International Conference on Object Oriented Information Systems, pp. 200–210. Springer, London, UK (2000)

Raymond, E.S.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media, Sebastopol (1999)

Segal, J., Morris, C.: Developing scientific software. IEEE Softw. **25**(4), 18–20 (2008). https://doi.org/10.1109/MS.2008.85

Segol, G.: Classic Groundwater Simulations Proving and Improving Numerical Models. Prentice-Hall, Amsterdam (1994)

Stallman, R.: The GNU project. (2018). https://www.gnu.org/gnu/thegnuproject.en.html, Accessed 12 Dec 2018

Stodden, V.: The legal framework for reproducible scientific research: Licensing and copyright. Comput. Sci. Eng. **11**(1), 35–40 (2009). https://doi.org/10.1109/MCSE.2009.19

Tapscott, D., Williams, A.D.: Wikinomics: How Mass Collaboration Changes Everything. Portfolio, London (2006)

Thacker, B.H., Doebling, S.W., Hemez, F.M., Anderson, M.C., Pepin, J.E., Rodriguez, E.A.: Concepts of model verification and validation. Tech. rep., Los Alamos National Lab., Los Alamos, NM (US) (2004)

Wikipedia: Code review—wikipedia, the free encyclopedia. (2018). https://en.wikipedia.org, [Online; Accessed 07 Dec 2018]

Will Schroeder: Why Open Source Will Rule Scientific Computing | The Kitware Blog. (2010). https://blog.kitware.com/why-open-source-will-rule-scientific-computing/

Wilson, G., Aruliah, D.A., Brown, C.T., Chue Hong, N.P., Davis, M., Guy, R.T., Haddock, S.H.D., Huff, K.D., Mitchell, I.M., Plumbley, M.D., Waugh, B., White, E.P., Wilson, P.: Best practices for scientific computing. PLOS Biol. **12**(1), 1–7 (2014). https://doi.org/10.1371/journal.pbio.1001745